

Georgia State University
ScholarWorks @ Georgia State University

Computer Science Dissertations

Department of Computer Science

12-5-2007

Distributed Web Service Coordination for Collaboration Applications and Biological Workflows

Janaka Lalith Balasooriya

Follow this and additional works at: https://scholarworks.gsu.edu/cs_diss

 Part of the [Computer Sciences Commons](#)

Recommended Citation

Balasooriya, Janaka Lalith, "Distributed Web Service Coordination for Collaboration Applications and Biological Workflows." Dissertation, Georgia State University, 2007.
https://scholarworks.gsu.edu/cs_diss/30

This Dissertation is brought to you for free and open access by the Department of Computer Science at ScholarWorks @ Georgia State University. It has been accepted for inclusion in Computer Science Dissertations by an authorized administrator of ScholarWorks @ Georgia State University. For more information, please contact scholarworks@gsu.edu.

DISTRIBUTED WEB SERVICE COORDINATION FOR COLLABORATIVE APPLICATIONS AND BIOLOGICAL WORKFLOWS

by

JANAKA BALASOORIYA

Under the Direction of Sushil K. Prasad

ABSTRACT

In this dissertation work, we have investigated the main research thrust of decentralized coordination of workflows over web services. To address distributed workflow coordination, first we have developed “Web Coordination Bonds” as a capable set of dependency modeling primitives that enable each web service to manage its own dependencies. Web bond primitives are as powerful as extended Petri nets and have sufficient modeling and expressive capabilities to model workflow dependencies. We have designed and prototyped our “Web Service Coordination Management Middleware” (WSCMM) system that enhances current web services infrastructure to accommodate web bond enabled web services. Finally, based on core concepts of web coordination bonds and WSCMM, we have developed the “BondFlow” system that allows easy configuration distributed coordination of workflows. The footprint of the BonFlow runtime is 24KB and the additional third party software packages, SOAP client and XML parser, account for 115KB.

INDEX WORDS: Web Services, Biological Workflows, Expressiveness, Formal Workflow Models, Distributed Coordination, Collaborative Applications

**DISTRIBUTED WEB SERVICE COORDINATION FOR COLLABORATIVE
APPLICATIONS AND BIOLOGICAL WORKFLOWS**

By

JANAKA BALASOORIYA

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree
Of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2006

Copyright By
Balasooriya Mudiyansele Janaka Lalith Balasooriya

2006

Distributed Web Service Coordination for Collaborative Applications and Biological Workflows

by

Janaka Balasooriya

Major Professor: Sushil K. Prasad
Committee: Shamkant Navathe
Rajashekar Sunderraman
Yi Pan

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2006

To my parents and family, for their guidance, support, love, and enthusiasm

Acknowledgements

A journey is easier when you travel together. Interdependence is certainly more valuable than independence. This thesis is the result of five years of work whereby I have been accompanied and supported by many people. It is a pleasant aspect that I have now the opportunity to express my gratitude for all of them. The first person I would like to thank is my supervisor Prof. Sushil K Prasad. I have been in his project since 2001 when I started my graduate studies at GSU. His overly enthusiasm and integral view on research and his mission for providing 'only high-quality work and not less', has made a deep impression on me. I owe him lots of gratitude for having me shown this way of research. Besides of being an excellent supervisor, he was as close as a relative and a good friend to me. I am really glad that I have come to get know him in my life. I would like express my sincere gratitude to Prof. Sham Navathe who kept an eye on the progress of my work, shared his wealth of knowledge and experience, and always was available when I needed his advise. I would also like to thank the other members of my PhD committee who monitored my work and took effort in reading and providing me with valuable comments on earlier versions of this thesis: Raj Sundarraman, Yi Pan, I thank you all. I had the pleasure to work with several two Master students, Mohini Pandhye and Jaimini Joshi who did their graduation work in our project and have been beneficial for the presented work in this thesis. I feel a deep sense of gratitude for my late father and mother who formed part of my vision and taught me the good things that really matter in life. The happy memory of my father still provides a persistent inspiration for my journey in this life. I am grateful for my sisters. I am glad to be one of them. I am very grateful for my wife, for her encouragement, love, and patience during the PhD period. One of the best experiences that we lived through in this period was the birth of our son Thilina, who provided an additional and joyful dimension to our life mission. The chain of my gratitude would be definitely incomplete if I would forget to thank the first cause of this chain, K.N.King and Unil Perera. Using Aristotle's words, The Prime Movers, my deepest and sincere gratitude for recommending me to the graduate program.

Table of Content

List of Figures.....	x
List of Tables	xiii
ABBREVIATIONS	xiv
CHAPTER 1 INTRODUCTION.....	1
1. 1 Motivation.....	3
1.2 Current State of the Art.....	5
1.3 Limitations of Current Technology	7
1.4 Problem Statement and Research Goals	12
1.5 Contributions and Significance.....	13
1.6 Organization of the Thesis	15
CHAPTER 2 BACKGROUND.....	18
2.1 Workflow Management Systems.....	18
2.2 Information Systems	27
2.3 Web Services	31
2.4 Merging Web Service and Workflows	35
CHAPTER 3 WEB COORDINATION BONDS	37
3.1 Introduction.....	37
3.2 Web Coordination Bond Concepts	39
3.2.1 Notations for Web Bonds.....	42
3.3 Evaluating Capabilities of Web Coordination Bonds.....	43
3.4 Modeling Power of Web Bonds.....	44
3. 5 Modeling Various Dependency Scenarios Using Web Coordination Bonds	50
3.5.1 Producer-Consumer Dependencies.....	50
3.5.2 Shared Resource Dependencies	52
3.5.3 An Application Scenario: Shared Calendars of Meeting Example.....	53
3.6 Related Work and Discussion.....	55
3.7 Summary	60
CHAPTER 4 EXPRESSIVNESS OF WEB COORDINATION BONDS	61
4.1 Modeling Workflow Control Flow Pattern: Background.....	62

4.1.1 Business Process Execution Language for Web Service (WS-BPEL)	65
5.1.2 Petri-net	66
4.2 Modeling Workflow Control Flow Patterns Using Web Coordination Bonds	67
4.2.1 Basic Control Flow Patterns	69
4.2.2 Advanced Synchronization Patterns	71
4.2.3 Patterns Involving Multiple Instances (MI)	77
4.2.4 State Based Patterns	82
4.2.5 Structural Patterns	86
4.2.6 Cancellation Patterns	88
4.3 Modeling Communication Patterns	90
4.3.1 Synchronous Communication	91
4.3.2 Asynchronous Communication	92
4.4 Related Work and Discussion	94
4.5 Summary	98
CHAPTER 5 WEB COORDINATION MANAGEMENT MIDDLEWARE	
SYSTEM	100
5.1 Limitations of Current Centralized Coordination	101
5.2 Evolution of Database Application Development	103
5.3 Functionalities Encapsulated by the Centralized Workflow	105
5.4 Web Service Coordination Management Middleware Architecture: An Overview	109
5.5 Web Service Coordination Management System	111
5.6 Web Service Management System	116
5.7 Summary	120
CHAPTER 6 SIMULATION AND VERIFICATION OF WEB SERVICE	
COORDINATION MANAGEMENT MIDDLEWARE	121
6.1 Realization of WSCMM Using Web Coordination Bonds	122
6.2 Background: Discrete Event System Specification (DEVS)	124
6.3 Simulating WSCMM Architecture	126
6.3.1 Message Handler	126
6.3.2 The Web Service Management System	129

6.3.3 The Web Service Coordination Management System (WSCMS).....	131
6.3.4 Web Service	132
6.4 Simulation Scenarios	134
6.4.1 Simulating Pre-Execution Dependencies.....	137
6.5 WSCMM: Compatibility with other Standards	141
6.6 Discussion and Related Work.....	144
CHAPTER 7 THE BONDFLOW SYSTEM	149
7.1 Limitations of Current Technology	152
7.2 The BondFlow Solution.....	152
7.3 Developer's View of BondFlow System	154
7.4 Two-Layered Workflow Software Architecture	156
7.4.1 Web Bond layer and the Bond Repository	160
7.4.2 Web Bond Layer	161
7.4.3 High-level Programmability	162
7.5 The BondFlow System Architecture: Design and Implementation	163
7.6 Handheld-Based Execution.....	167
7.7 System Evaluation	170
7.8 Related Work and Discussion.....	176
7.9 Summary	178
CHAPTER 8 BIOLOGICAL WORKFLOWS	179
8.1 Challenges in Biological and Data and Tool Integration.....	179
8.1.1 Web service Enabled Biological Tools.....	181
8.2 Motivating Example.....	185
8.3 Using the BondFlow System for Biological Workflows	188
8.3.1 Workflow Development Methodology	190
8.3.2 Alignment Region Comparison Workflow using the BondFlow System.....	192
8.3.3 System output.....	192
8.4 Conclusions and Future Work	197

CHAPTER 9 CONCLUSIONS AND FUTURE WORK	199
9.1 A Platform to Configure and Deploy Distributed Workflows over Web Services.....	200
9.2 Future Work	201
BIBLIOGRAPHY	203

List of Figures

Figure 1.1 Purchase Order Workflow	5
Figure 1.2: Current State of the Art of Web Service Workflow Development:	
Architecture of traditional WS-BPEL Implementation	6
Figure 2.1: Manual Supply Chain Management	19
Figure 2.2: Automated Supply Chain Management	20
Figure 2.3: Quotation Process Workflow Specification	21
Figure 2.4: Interaction among Workflow Activities	23
Figure 2.5: Workflow Reference Model	24
Figure 2.6: A Typical Workflow Engine	25
Figure 2.7: A Typical Information System	27
Figure 2.8: B2B Integration	29
Figure 2.9: The Interoperability Problem	30
Figure 2.10: Web Services: A Uniform Interface and a Common Communication	
Protocol	31
Figure 2.11: Web Service Definition	31
Figure 2.12: Current Web Services Infrastructure	32
Figure 2.13: Web Service Composition	33
Figure 2.14: Taxonomy of Web Service Standards	34
Figure 2.15: Workflow Language and Protocol Chronology	35
Figure 3.1: Analogy between Chemical Bonds and Web Bonds	38
Figure 3.2: Subscription bond	42
Figure 3.3 Negotiation bond	42
Figure 3.4 Subscription-negotiation bond pair	43
Figure 3.5: Petri Net with inhibitor arcs (EPN)	47
Figure 3.6: Simulating EPN using web bonds	47
Figure 3.7: Coordinating Producer-Consumer web Processes	50
Figure 3.8 Modelling Resource Sharing among Competing Web Processes	52
Figure 3.9 Coordinating multiple producers with a consumer Web process	53
Figure 3.10: Scheduled Meeting	54

Figure 3.11: A Tentative Meeting.....	55
Figure 4.1: Petri-net model	66
Figure 4.2: Parallel Split	69
Figure 4.3: Simple Merge	70
Figure 4.4: Advanced Synchronization.....	71
Figure 4.5:Synchronization Pattern	73
Figure 4.6: Multi Merge.....	74
Figure 4.7: Discriminator pattern.....	76
Figure 4.8: MI without Synchronization.....	78
Figure 4.9: MI with prior design time knowledge	79
Figure 4.10: Differed Choice	83
Figure 4.11: Mile stone pattern.....	84
Figure 4.12: Interleaved Parallel Routing.....	85
Figure 4.13: Arbitrary cycle.....	86
Figure 4.14: Arbitrary cycle using web bonds.....	87
Figure 4.15: Reply/Request	91
Figure 4.16: Publish-Subscribe Communication	93
Figure 5.1: Current State of the Art: Composite Web Process as a Central Coordinator.....	101
Figure 5.2: Evolution of Database Application Infrastructure	104
Figure 5.3: Proposed development for web service infrastructure	105
Figure 5.4: Functional decomposition of composite web process.....	107
Figure 5.5: Web Service Coordination Middleware Overview	109
Figure 5.6: Enforcing Pre Execution Dependencies	112
Figure 5.7: Enforcing Post Execution Dependencies	114
Figure 5.8: Web service management System.....	116
Figure 5.9: Coordinator Proxy Object Architecture	117
Figure 5.10: Typical Flow within a coordinator proxy object.....	118
Figure 6.1: Enforcing Dependencies Using Web Coordination Bonds	122
Figure 6.2: Simulation Scenario	124
Figure 6.3: DEVS simulation model.....	125
Figure 6.4 : WSCMM Simulation Model	127

Figure 6.5: Message Handler	129
Figure 6.6: Web Service Management System.....	130
Figure 6.7: Web Service Coordination Management System.....	131
Figure 6.8: Simulation Architecture	134
Figure 6.9: Message Routing in WSCMM Simulation.....	136
Figure 7.1 Flow within the Coordinator Object.....	154
Figure 7.2: Developers Perspective of the BondFlow System	155
Figure 7.3: Two-Layer Workflow Software Architecture	157
Figure 7.4: Web Service Coordinator Proxy Object.....	158
Figure 7.5: Flow within a Proxy Object.....	159
Figure 7.6: Elements of a Typical “Bond”	160
Figure 7.7: Sample Bond Repository.....	160
Figure 7.8: Purchase Order Workflow.....	162
Figure 7.9: BondFlow System Architecture	164
Figure 7.10: Proxy object generation.....	164
Figure 7.11: The BondFlow Runtime	166
Figure 7.12: Workflow configuration.....	166
Figure 7.13: Workflow Distributed among Several iPAQ’s.....	169
Figure 7.14: Book Price Workflow.....	172
Figure 7.15: Traffic Condition Workflow	172
Figure 7.16: Purchase order workflow	173
Figure 7.17: Online book purchase workflow	173
Figure 7.18: Execution timings for sample workflow control flow patterns.....	175
Figure 8.1: Alignment Region Comparison Workflow	186
Figure 8.2: The BondFlow system for Biological Workflows	188
Figure 8.3: The BondFlow System: Users Perspective	191
Figure 8.4 : Alignment Region Comparison Workflow using web coordination bonds	193
Figure 8.5: The BondFlow System Executing Alignment Region Comparison Workflow	194
Figure 9.1: Workflow Coordination Architectures of the BondFlow System	201

List of Tables

Table 3.1 Comparison of Web Service Coordination Languages/Standards.....	57
Table 4.1 Workflow Control Flow Patterns.....	62
Table 4.2: BPEL Primitives	65
Table 4.3 Support for workflow control patterns in different web service composition languages and standards	68
Table 4.4: Patterns Involving Multiple Instances	81
Table 4.5: Cancellation Patterns	89
Table 4.6: Communication patterns	90
Table 6.1: External Messages among Web Services When Enforcing Dependencies Using Web Coordination Bonds.....	123
Table 6.2: Message tag and the outgoing message ports at the Message Handler	128
Table 6.3: Actions taken at WSMS.....	131
Table 6.4: Actions Taken at WSCMS.....	132
Table 6.5: Different states of Middleware Components.....	133
Table 6.6: Simulation Output for Incoming Messages	137
Table 6.7: Architectural Enhancements to Web Services.....	145
Table 7.1: Size of WSDL (number of methods) vs. Proxy Object Generation Time	171
Table 7.2: Workflow execution timings	174
Table 7.3: Footprint of the workflow.....	174

ABBREVIATIONS

ASAP	Asynchronous Service Access Protocol
BPEL4WS	Business Process Execution Language for Web Services
WS-BPEL	
BPMI	Business Process Modeling Initiative
BPML	Business Process Modeling Language
BPMN	Business Process Modeling Notation
BPSS	Business Process Specification Schema
CFA	Communicating Finite Automata
CPO	Coordinator Proxy Object
DAML-S	DARPA Agent Markup Language
DBMS	Database Management System
DEVS	Discrete Event System Specification
HTTP	Hypertext Markup Language
IDE	Interactive Development Environment
NASL	Network Access Service Language
OASIS	Organization for the Advancement of Structured Information Standards
QoS	Quality of Service
RDF	Resource Description Framework
REST	Representational State Transfer
RPC	Remote Procedure Call
SOC	Service Oriented Computing
SOAP	Simple Object Access Protocol
UDDI	Universal Description, Discovery and Integration
UML-WSC	UML Profile for Web Service Composition
URI	Universal Resource Locator
W3C	World Wide Web Consortium
WDS	Well-Defined Service
WFMS	Workflow Management System
WS	Web Service
WSCI	Web Service Choreography Interface
WSCMS	Web Service Coordination Management System
WS-Conversion	Web Service Conversation
WS-Coordination	Web Service Coordination
WSDL	Web Service Description Language
WSFL	Web Service Flow Language
WSMF	Web Service Modeling Framework
WSMS	Web Service Management System
WS-Policy	Web Service Policy
WSs	Web Services
WSTMW	Web Service Transaction Middleware
WS-Transaction	Web Service Transaction
WWW	World Wide Web
XL	XML Language for Web Service Composition

XLANG	Web Services for Business Process Design
XML	Extensible Markup Language
YAML	Yet Another Workflow Model
WSCMM	Service Coordination Management Middleware

CHAPTER 1

INTRODUCTION

“Software as a Service” or Service Oriented Computing (SOC) is the recent notable development in software engineering [Alo04, Pap05, Wee05]. These software services will be running on heterogeneous platforms and distributed information networks, providing services to other entities in the network [Gir04]. Web service (WS) infrastructure is arguably the most important realization of the SOC architecture [Wee05]. Web service is defined as a “self-contained modular application that can be described, published, located, and invoked over the net” [Ley02]. It encapsulates the computational complexity and hides system and network heterogeneity. Web services expose their functionality through a well-defined interface. Client entities interact with the interface of the web services. One can harness the true potential of the WS infrastructure by integrating different web services together to form sophisticated applications such as workflows [Dus04, Ko03]. Therefore, in the SOC model, WSs become the building blocks based on which new applications are created. Such integration enables inter-organizational collaboration and spans application domains as diverse as enterprise e-commerce applications (supply chains, work flows, and virtual organizations) [Dus04]), personal applications (travel, calendaring and scheduling) [Moo04], and scientific biomedical applications (biomedical data and tool integration, and workflows) [Ber03, Nek03, Moo04, Pic99, 80, Wil00, Aal02].

However, the current state of the art in developing such workflow applications over web services employs a centralized composite process to coordinate the constituent web services. This coordinator process is complex, less scalable, and bulky. Moreover,

currently there is no fundamental framework for workflow dependency modeling. Therefore, currently, workflow development process is a tedious task and confined only to expert developers.

In this dissertation, we have investigated the main research thrust of decentralized coordination of workflows over web services and its applicability biological workflows. First, we have proposed and formally investigated “web coordination bonds” as a capable set of primitives for distributed workflows over web services. Then, we have designed and prototyped our “*Web Service Coordination Management Middleware (WSCMM)*” that enhances current web service infrastructure so that web services become stateful self-coordinating entities enabling them to actively participate in workflows. Finally, based on the core concepts of web coordination bonds and WSCMM, we have developed our “*BondFlow*” system that allows easy configuration and distributed deployment of workflows over web services. The BondFlow system distributes the centralized coordination logic by (i) extending the web services into self-coordinating entities using coordinator proxy objects, and (ii) creating the workflow over these entities by interconnecting them into a distributed network of objects using web bond primitives. Finally, we have employed the BondFlow system to configure and execute biological workflows such as DNA sequence analysis.

Chapter 1 starts with motivation for this research using few web service based workflow examples. Then, we discuss limitations of the current state of the art in developing workflows over web services. Next, we state our research goals and contributions. Finally, we highlight the organization of this dissertation.

1. 1 Motivation

Following examples of a few selected web services based applications highlight the corresponding research domains.

E-Commerce Applications: E-commerce applications rapidly change the way businesses perform their transactions. However, as most researchers have pointed out, “real revolution comes when businesses begin to conduct their activities electronically with other businesses over the web thereby increasing efficiency (higher throughput) and robustness (easy modification, correctness verification)” [Sha01]. For example, in a supply chain application scenario, we can envision that a consumer’s web service automatically finds a suitable supplier and places the order using pre-specified rules/logic and business relationships. The intermediate steps may be as follows. The consumer calls for bids. Each potential bidder’s web service evaluates requirements of the buyer and subsequently enters the bidding process. Then the buyer’s web service evaluates the bids, selects a supplier and places the order. Finally, the suppliers web service contacts the transportation service for delivery. Development of such complex applications is a tedious task today. However, a suitable workflow infrastructure can automate this process.

Travel Applications: Future web services will much more sophisticated, interconnected, and interoperable [Har04]. For example a travel application integrates reservations of flights, rental cars, and hotel accommodations. Most existing travel reservation applications do not combine and maintain a global relation among these services. As a result, manual changes need to be performed if one portion of the itinerary changes. The process behind such applications would not only integrate these web services, but also

enforce Quality of Service (QoS) constraints such as deadlines and budget requirements. If the flight is cancelled, then automatic cancellation of car and hotel reservations will be triggered, thus easing the burden on the user to manually cancel all associated reservations.

Bio-Medical Applications: Rapid development of ad-hoc and other collaborative applications by leveraging off existing bio-medical web services will be the key to bring the Internet's collaborative potential to the non computer scientists. These bio-medical web services would comprise various heterogeneous and autonomous data stores as well as a myriad of higher-level value-added bio-informatics server applications (e.g., search and data mining engines, genetic databases, molecular dynamics tools, pattern recognizers, and algorithmic tools) published as web services [Nav04]. Scientists must be empowered to easily and rapidly compose and link existing bio-medical web services to create ad-hoc client as well as server applications. For example, such capabilities would be needed to quickly put together an experiment-specific ad-hoc application for recognizing protein molecules with certain descriptors by accessing a select set of biochemical databases, passing the aggregated results to a simulated annealing tool, and inserting a bit of scientific logic which has evolved from experimentation [Byr01].

Many of those applications are long-running transactions and workflows that require much more beyond the currently supported invoke/respond protocols [Dou03, Dou03, Lit03]. Thus, efficient coordination technologies are required to rapidly develop and deploy robust collaborative applications by leveraging off the existing web services [Pic99, Jon03]. Therefore, the underlying computational issues are fundamental and with wide scope.

1.2 Current State of the Art

Figure 1.1 illustrates the purchase order workflow presented in the WS-BPEL (Web Services Business Process Execution Language) specification [Wan05]. The operation of this workflow is as follows: on receiving a purchase order, the *receive purchase order* web service processes the request and trigger three concurrent tasks to initiate the price calculation, select a suitable shipper, and schedule the production and shipment. Once all three tasks are done, invoice processing task will be initiated.

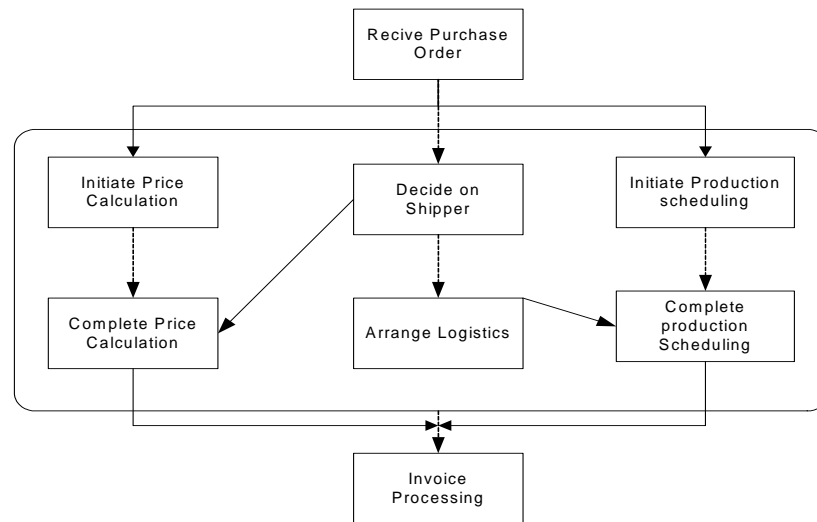


Figure 1.1 Purchase Order Workflow (dark arrows represent the control flow dependencies while dashed arrows represent data flow dependencies)

Figure 1.2 illustrates the software architecture of the WS-BPEL based implementation of the workflow. It models the composite workflow process as a separate state-preserving

web process encapsulating all the data flow and control flow requirements. This is due to the fact that WSs have been designed to be stateless autonomous entities. Thus, they are not active participants in the workflow. A composed web process needs to encapsulate numerous functionalities ranging from application logic to transaction management. It is the designer's responsibility to focus on low level (atomic) details such as message correlation, and state (context) information to high-level application logic. Therefore, BPEL is at the level of the assembly language for web service composition and coordination. Moreover, the composite web process becomes a central coordinating entity. Section 1.3 further elaborates limitations of the current technology.

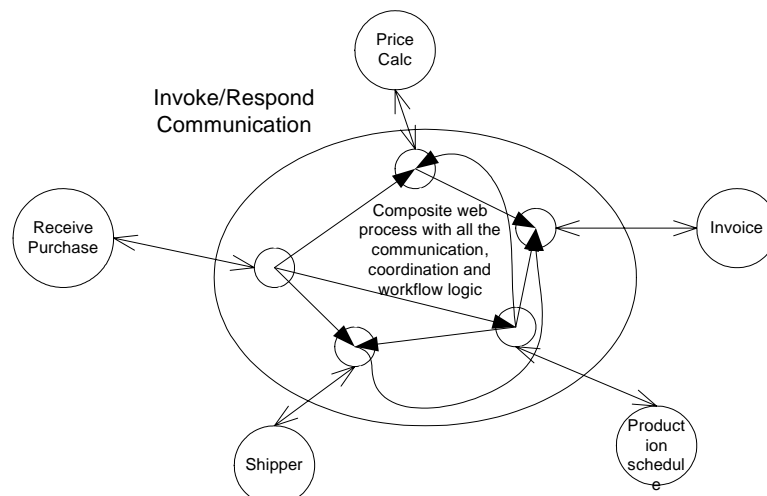


Figure 1.2: Current State of the Art of Web Service Workflow Development:

Architecture of traditional WS-BPEL Implementation

1.3 Limitations of Current Technology

This section highlights limitations on workflow coordination architecture, coordination technologies, and deployment and execution platforms of the current web service workflow technology.

Workflow Coordination: As we have seen in the purchase order workflow example, current state of the art in developing workflows over web services is to model the composite web service (workflow process) as a separate state-preserving web process, as WSs are stateless and not active participants. Thus, the composed web process needs to encapsulate numerous functionalities ranging from application logic to transaction management making it a central coordinating agent [Alo04, Bar05]. However, centralized coordination is not desirable in highly distributed web services infrastructure: (i) Due to security, privacy, or licensing imperatives, some web-based objects will only allow direct pair-wise interactions without any coordinating third-party entity; and (ii) Centralized coordination/workflows suffer from issues such as scalability, performance, and fault tolerance [Gir04]. Efforts such as IBM symphony [Gir04] try to eliminate centralize coordination by partitioning centralized BPEL code into separate modules so that they can run in a distributed setting. However, there are limitations to such efforts. First, it is necessary to develop the centralized BPEL code and then distribute it. Second, there are usually problems in partitioning the code in complex application scenarios such as long running transactional applications without proper infrastructure support. Middleware platforms for web services are emerging as a solution to this problem.

In [Bar05] authors point out that it is necessary to identify different levels of software abstractions (viewpoints) from web service composition and coordination and generalize them. These generalized functionalities can be used to further enhance web service's interface. This will transform the web services we know today into coordination aware stateful web entities making the application development less programming intensive and enabling distributed coordination. While investigating the current efforts towards this goal, it is interesting and encouraging to see that a significant effort is being made in both academia and industry [Bar05, Sch05, Ben05].

Coordination primitives: Unavailability of a comprehensive fundamental framework to model workflows is another significant issue in current workflow development. There are many overlapping and competing languages for web service workflow development. However, none of them are comprehensive enough.

In [Gri01], authors propose use of Petri nets for web service workflows. Petri nets are a well-founded process modeling technique with formal semantics [Aal02]. They have been used to model and analyze several types of processes including protocols, manufacturing systems, and business processes [Gri01]. BPEL4WS is becoming popular in web services community as a workflow language. BPEL allows a mixture of block and graph structured process models, thus making the language expressive at the price of being complex [Aal03b]. SUN, BEA, SAP and Intalio came up with another standard called WSCI (Web Service Choreography Interface). BPML and ebXML are other candidates in the same race. WS-Coordination (Web Services Coordination) is a proposed IT industry standard, which contains specification for composition and

coordination among distributed web services [Woh03]. The PhD thesis presented in [Kie02] has studied the expressiveness and suitability of these languages for modeling workflow control flow patterns. Using those workflow patterns as a benchmark, web services composition and workflow languages such as BPEL4WS, XLANG, WSFL, BPML, WSCI, and High-level Petri-net-based languages have been evaluated [Aal03b]. This evaluation shows that none of these languages are compressive enough to model workflow dependencies. This abundant number of languages/standards has still failed to give a framework, which is fundamentally sound and yet powerful in operation. To overcome this problem, initially, a critical evaluation of these standards is required

Workflow Deployment and Execution Platforms: World Wide Web became so popular due to its simplicity and easy accessibility. In contrast, CORBA, RMI and DECOM did not succeed to the level that their proponents expected. This is mainly due to the complexity of these technologies despite great features they carry [Dus04, Woh03]. Web services are to bridge the gap between two technologies. Therefore, ideally, applications that we configure using web services should be able to deploy and execute in web-like (preferably over Internet) infrastructure enabling them to be executed on both wired and wireless devices including servers, PCs, handhelds, and even on cell phones. Executing workflows over wireless devices has significant benefits [Dus04-Haw05]. Portions of long-running workflows can reside on handheld device providing monitoring and controlling capabilities as well as hosting services. Current web service workflow deployment platforms are difficult to interact with and confined only to expert users. Additionally, current platforms consume significant amount of resources and are difficult

to deploy on limited resource wireless devices. Some of the current web service composition and coordination architectures inherently assume that services are resident on the wired infrastructure. However, there is an increasing interest in both industry and academia to empower mobile devices. In [Cha04], authors describe issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition. A distributed architecture and associated protocols for service composition in mobile environments that take into consideration mobility, dynamic changing service topology and device resources are presented in [Haw05]. The composition protocols are based on distributed brokerage mechanisms and utilize a distributed service discovery process over ad-hoc network connectivity. In [Dus04] authors present architecture for mobile device collaboration using web services. In [Mna04] authors present a rapid application development environment for mobile web services. In [Ste03, Haw05] authors present web service based mobile application integration frameworks. However, most of these technologies consider handheld devices as clients.

Biological data and tool integration: Enormous amount of biological data is being produced by biologists. It is estimated that about one billion data stores available. These data inherits heterogeneity in their format and representation [Lab03]. Also, numerous applications have been developed to analyses these data and produce meaningful results such as identification unknown species and diagnosis of deceases. Data analysis requires multiple resources to be integrated and filter data from one source and feed into another. However, these data sources are heterogeneous in nature. For example,

- i) data being stored are highly diverse
- ii) data being stored are highly representational heterogeneous
- iii) data sources and tools are autonomous and have different interfaces and querying capabilities.

In [Pao05, Lab03], authors discuss aforementioned facts in detail. Currently, most popular and highly used methodology is to develop programs (or scripts) from the scratch. This is very time consuming and ineffective. In [Her04], authors state that manual data processing has been pushed to the limit and requires more pragmatic approaches. As a solution web browser based data processing tools have been developed. Research groups at European institute of Bioinformatics have identified significant drawbacks in this approach.

- i) Web browser based tools are difficult to use in case of large amount of data to be retrieved and analyzed.
- ii) In case of workflow applications, the developer needs to copy data from one source and paste them to the other interface. This is tedious, as data formatting and copy and paste process need to be repeated several times.

Researchers have identified web services are a better technology to deal with these difficulties. Data producers can have unique interfaces to supply data and users can use standard set of technologies to access them. Also, large amount of data can be attached as SOAP attachments and feeding from one source to another can be automated relatively easily [Lab03]. Many major bioinformatics institutes such as NCBI, DDBJ, and EBI have already started converting their biological data sources and search tool into web services.

However, still the workflow development and deployment platforms are difficult to use and need significant amount of programming. For example, Pegasys [Sha04], Taverna[Hul06], and Discovery Net[DiscoveryNet] are great systems with graphical user interfaces to compose biological workflow. However, most of them are domain specific and suitable for pre-configured systems and workflows. For example, Pegasys [Sha04] system has been designed to achieve three goals: modularity, flexibility, and data integration in biological workflows. It includes tools for pair-wise and multiple sequence alignment and gene prediction, RNA gene detection. Users of the Pegasys system create a DAG to represent the workflow. Each node v , represents either a input sequence, an individual program or a output node while an edge between two nodes represents the data flow. DAG can be created dynamically at runtime using the Pegasys GUI and subsequently converted into a structured XML file, which will be transferred to the Pegasys server that executes the workflow. Data filters take care of input/output formatting from one tool to another. However, one major draw back of the system is that programs and filters need to be written by experts to add new tools to the system. System is pre-configured.

1.4 Problem Statement and Research Goals

In this dissertation we have undertaken following research thrusts to tackle outstanding issues identified in the preceding section.

1. Develop a core set of capable primitives which enable Web services to hook together in a desired structure to enforce automatic information flow, group

constraint satisfaction, and data and control dependencies, all without any central coordinating authority. Prove expressive power, analytical power, and sufficiency of web coordination bonds.

2. Extend the web service infrastructure beyond the basic service architecture (invoke and respond) to self-coordinating web processes collaborating among themselves in the desired configuration as per user's application (transient to long lasting).
3. Create an easy to use platform so that developers including non-computer scientists can configure and deploy their workflows.
4. Evaluate the performance and capabilities of the prototype "BondFlow" system .
5. Apply the BondFlow system to develop biological data and tool integration.

We have made following contributions in this dissertation work.

1.5 Contributions and Significance

1. *Set of Coordination Primitives for Workflow Dependency Modeling:* We have developed "Web Coordination Bonds" as a capable set of primitives for distributed workflow coordination over web services. Web bond primitives are as

powerful the Petri nets extended with inhibitor arcs^{*} and have sufficient modeling and expressive capabilities to model workflow dependencies.

2. *Distributed Coordination:* We have designed and prototyped our Web Service Coordination Management Middleware (WSCMM) system that enhances current web services infrastructure. WSCMM distributes the workflow coordination among participant web services by generating an “intelligent” web service Coordinator Proxy Object (CPO) or coordinator object for short per web service. These coordinator objects are stateful and enable encapsulated web services to be interconnected. An interconnected coordinator object together with its dependency parameters represents a coordination aware workflow node on behalf of the encapsulated web service. This transforms current stateless passive web services into self-coordinating active workflow entities.

3. *The BondFlow System :* Based on core concepts of web coordination and WSCMM, we have developed the Bondflow system that allows easy configurability and distributed workflow coordination. Also, the footprint of the BondFlow runtime is 24KB and the additional third party software packages, SOAP client and XML parser, account for 115KB. Moreover, the footprint of the coordinator object is small (~10KB) enabling them to reside on java-enable handheld devices.

^{*} Thereafter, for the convenience, we will refer to the Petri nets extended with inhibitor arcs as “extended Petri nets.”

4. *Using the BondFlow System for Biological Workflows:* We developed few biological workflows such as Alignment Region Comparison Workflow using the BondFlow system. We further layout a stepwise methodology to develop simple biological workflows using this system. Steps involve (a) finding biological data sources and tools, and wrapping them into web services; (b) generating data adaptor web services for each connector edge in the ad-hoc workflow; (c) configure the workflow over web-enabled tools, data sources, and data adaptors; (d) execute workflow. Currently, the first step is in research state and step two is function is cases where data input and output requirements are specified using regular expressions.

1.6 Organization of the Thesis

The remainder of this dissertation is organized as follows.

Chapter 2 introduces the reader to technologies such as workflows, web services, and workflows over web services. The main purpose of this chapter is to provide readers with required understanding of the key technologies that help them to follow the remaining chapters smoothly.

Chapter 3 introduces the idea of web coordination bonds as a capable set of primitives for distributed workflow dependency modeling. Also, we establish that web bonds are as powerful as extended Petri nets (Petri net with inhibitor arcs) in their modeling power. We also illustrate the expressive capabilities of web bonds by modeling various dependency scenarios.

Chapter 4 further elaborates the expressiveness of web bonds by modeling a comprehensive set of workflow control flow patterns and distributed communication patterns. None of the current coordination technologies are capable of comprehensively modeling them. This exercise proves that web bonds are superior to the current technology in terms of their expressiveness and modeling of complex coordination.

Chapter 5 discusses the architectural enhancements that are needed to distribute the workflow coordination among web services. In this chapter we have undertaken the task of architecting our WSCMM. First, we have identified major functionalities encapsulated by the current web service workflows. Then we formulate the architecture of the middleware system to encapsulate generic layers of functionality.

Chapter 6 discusses the realization of WSCMM using web coordination bonds. Then we simulate the architecture for correctness verification. Discrete Event System Specification (DEVS) simulation tool has been used for the simulation. We have simulated web bond interactions and a simple workflow scenario. Our simulation results show that the middleware behaves accurately.

Chapter 7 discusses our prototype implementation of the BondFlow system. The BondFlow system is based on web coordination bond and WSCMM concepts. It provides an environment to easily configure and execute distributed workflows over web services. The workflow deployment environment is light weight and can be used to deploy workflows on small handheld devices.

Chapter 8 illustrates the use of the BondFlow system in biological workflows. First we identify issues related to biological workflows creation (data and tool integration). Then, we discuss the use of web service technology in biological data and tool integration. Finally, we illustrate the configuration and deployment of DNA Alignment Region Comparison using the BondFlow system.

Finally, Chapter 9 presents conclusions of this dissertation work and future research directions.

CHAPTER 2

BACKGROUND

In Chapter 1, workflow applications over web service have been identified as one of the major trends in developing current Internet applications. The four main research thrusts performed in this dissertation research work are aimed at finding a better technology for distributed workflows over web services. Thus, the content of this dissertation is based on two technologies, workflows and web services. This chapter is devoted to give a sufficient understanding of the two technologies and discuss how these two technologies have merged. We present this as a historical perspective as well as a technical background that helps readers better absorb the contributions of this work. We do not focus on any specific technology but will provide a comprehensive discussion on the technology behind workflows and web services. The remaining chapters discuss specific technologies wherever applicable. First, we discuss workflow management systems and problems they have faced in Enterprise Application Integration (EAI). Then, we present technological reasons that are contributing to the advancements needed in web service technology. Finally, we discuss how these two technologies have merged (or will be merged) together.

2.1 Workflow Management Systems

The origin of (Workflow Management System) WfMSs is *office automation*. Office automation can be simply described as routing electronic version of administrative documents such as project reviews from one point to another. Initially, office automation was email based and later it is incorporating more sophisticated web-based forms. In

web-based forms, most of the document/information routing and decision-making have been automated. The sequence of such actions takes place to complete one task. This kind of composition of tasks is called *administrative workflows*.

In industry, activities are not as simple as document routing. Initially, significant portion of human involvement was needed in the process. For example, earlier, supply chain applications had been handled manually where significant portions of workflow activities were manual activities (Figure 2.1). However, many of these tasks such as order processing, shipping management, and quotation can be automated. Such a automated process is capable of handling complex business actives as oppose to simple document management. As technology matured, manual handling of many activities have been transferred to software applications and tools. Figure 2.2 illustrates the automation of supply chain management. It is clear that tools/software modules interact with each other in a given order to accomplish the task. Such applications are called *production workflows*.

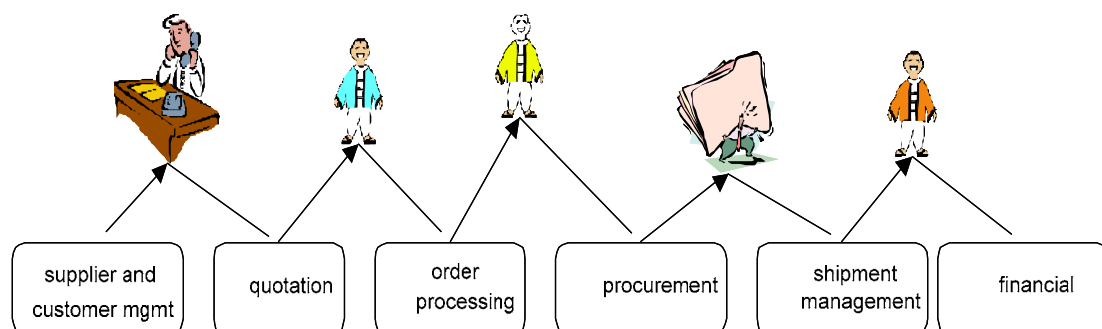
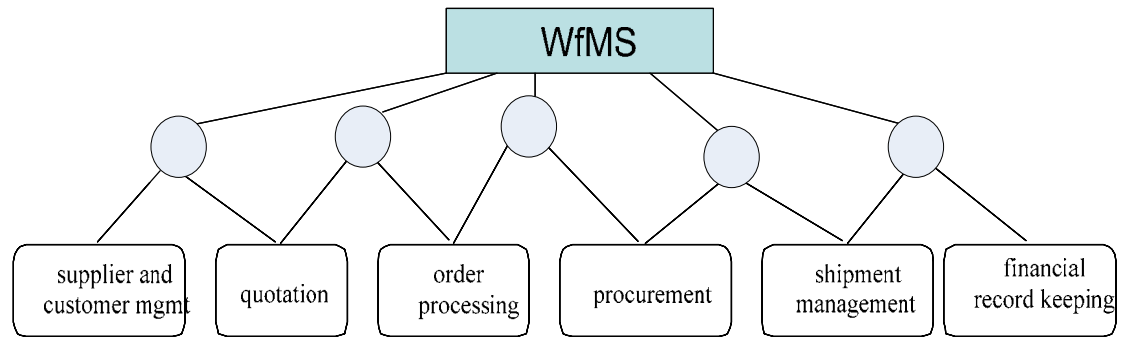


Figure 2.1: Manual Supply Chain Management [9]



(WfMS: Workflow Management Systems)

Figure 2.2: Automated Supply Chain Management [Alo04]

Characteristics Workflow Applications: Software applications interact in different ways to enforce the workflow requirements. These interactions can be triggered by manual entities or can be automated. However, in order to maintain the correct behavior of the workflows, such systems need to have an administrator. This administrator in its configuration is called the *workflow management system*. Workflow management system makes sure that the order of execution of activities is correct and handles any errors (Figure 2.2) and exceptions rising during the execution.

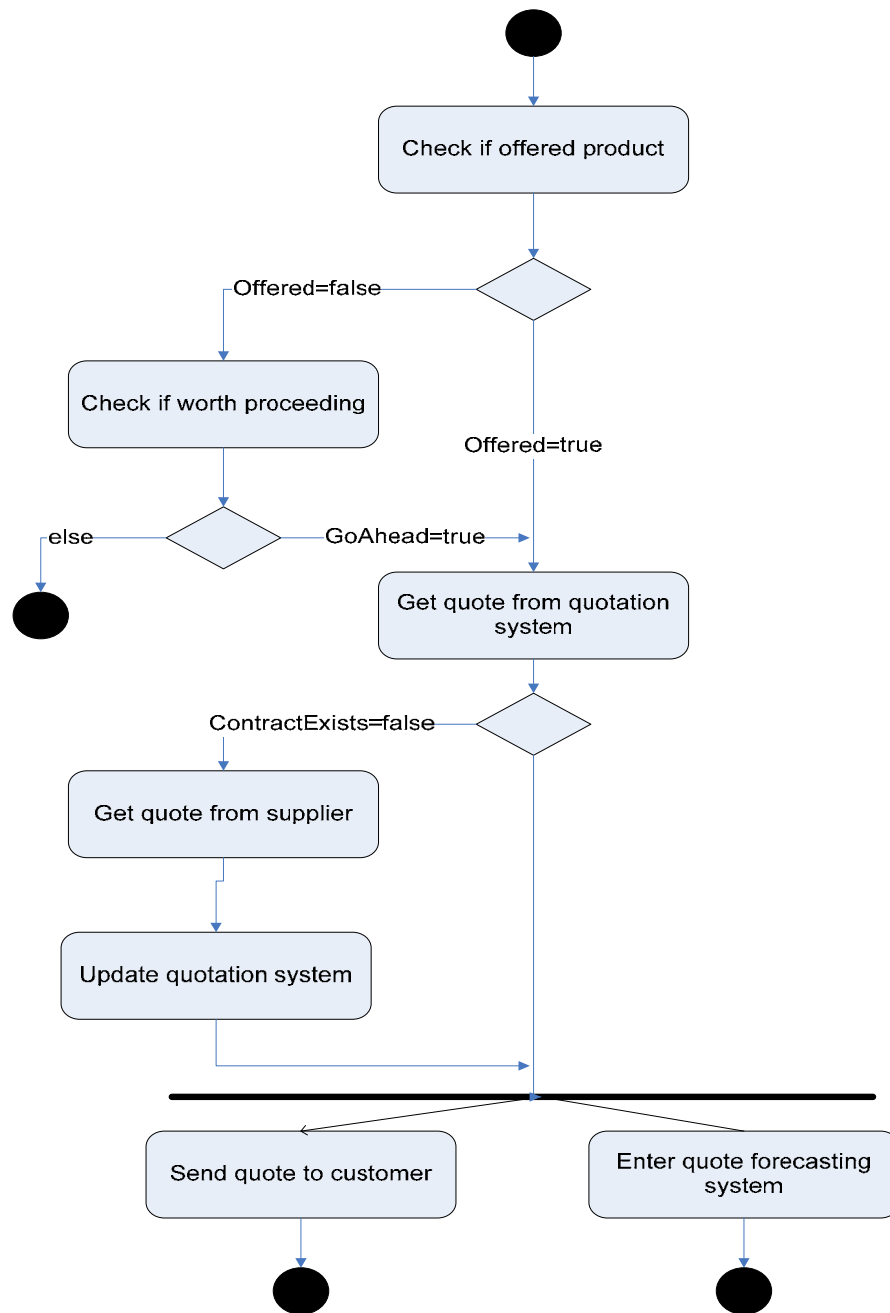


Figure 2.3: Quotation Process Workflow Specification [Alo04]

Figure 2.3 illustrates the workflow process to model the quotation process. Here, rectangular boxes represent processing elements while diamond shapes represent decision-making elements. For example, “get quote from quotation system” process

can be started only if one of the variables, “GoAhead” or “offered” is true. These variables essentially carry the control. Thus, “get quote from quotation system” has *control dependencies* and in this case it is an “OR” join. Similarly, it needs data to process the order. These types of dependencies are called *data dependencies*. Moreover, after the “update quotation system” process, it has to split the control into two paths (customer and the quote forecasting system). Many such data and control dependencies can occur in a workflow [Aal03a, Aal03b]. In this dissertation we will study these dependencies in detail [Chapter 3, Chapter4]. It is the responsibility of the workflow management system to integrate different activities together to enforce such workflow dependencies.

One of the major issues in such workflow applications is to integrate different system (s) together. This is mainly because these systems are running on different platforms and maintained by different departments. Any integration and process automation implies bringing all participating autonomous, heterogeneous entities together. Typically, workflow activities interact through message brokers where network and system heterogeneities are handled using adaptors (Figure 2.4). WfMSs focus on the definition and maintenance of the integration logic of such systems.

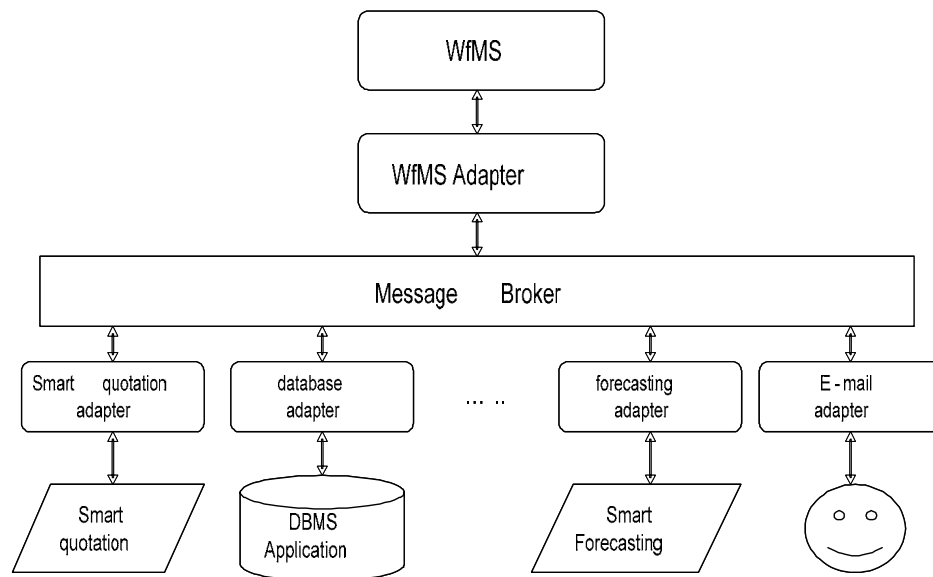


Figure 2.4: Interaction among Workflow Activities [Alo04]

Thus, workflow is a complex process with many requirements including workflow definition, dependency modeling, error handling, and modeling inter-operation among activities. In 1996, the Workflow Management Coalition (WfMC (<http://www.wfmc.org/>)) was formed to standardize the workflow activity definition and their requirements. WfMC defines the workflow as follows.

The Workflow Definition: *“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” [WFMC].*

This definition captures the essence of a workflow process. There are significant points that we can comprehend from this definition. First, work is fully or partially *automated* thereby information is being passed from one activity to another electronically and

decision are being made without (or with minimum) human interaction. Also, there are a set of *rules* that govern the behavior of the workflow. In order to accommodate these requirements of this definition, WfMC has defined a workflow reference model. The workflow reference model essentially specifies a framework for workflow systems identifying their characteristics, functions, and interfaces. It defines five interfaces as shown in Figure 2.5. Here, we will briefly review each interface.

Workflow Reference Model

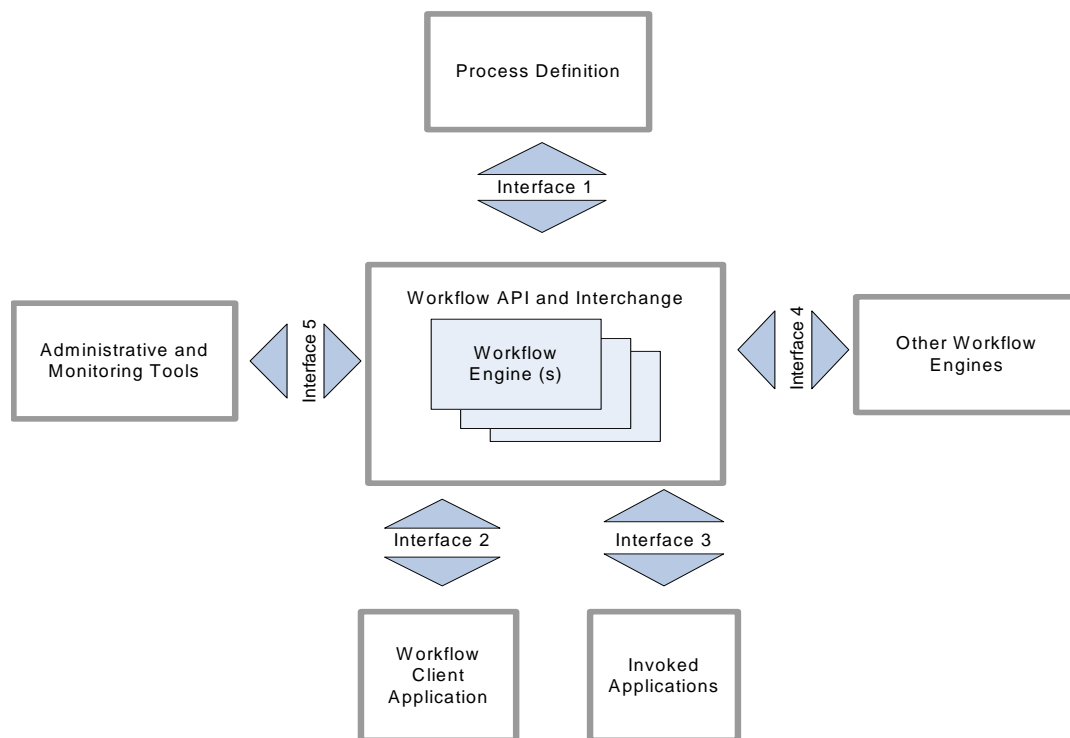


Figure 2.5: Workflow Reference Model [WfMC]

Process Definition Tools Interface: Process definition is the task of modeling control flow, data flow, and other dependencies of a workflow. In other words, the workflow process defines the relationship among the activities of a workflow. Workflow

description languages such as XPD, WS-BPEL and BPML support modeling these relationships (dependencies). The workflow engine interprets the workflow definition and enforces these dependencies. In Chapter 3 and 4, we extensively look at different (web service) workflow languages. These chapters also discuss web coordination bonds, one of the significant outcomes of this dissertation work, as a mechanism to define workflow processes.

The Workflow Engine: The main task of the workflow engine is to retrieve the workflow definition, determine which node (activity) is to be processed, acquire required resource(s), and place them in the work queue. Figure 2.6 illustrates the components of a generic workflow engine. The Inbound queue returns data/control from completed tasks. Based on that it determines the next task to be executed. In Chapters 5, 6, and 7 we discuss workflow enactment in our BondFlow system, which is based on web coordination bonds.

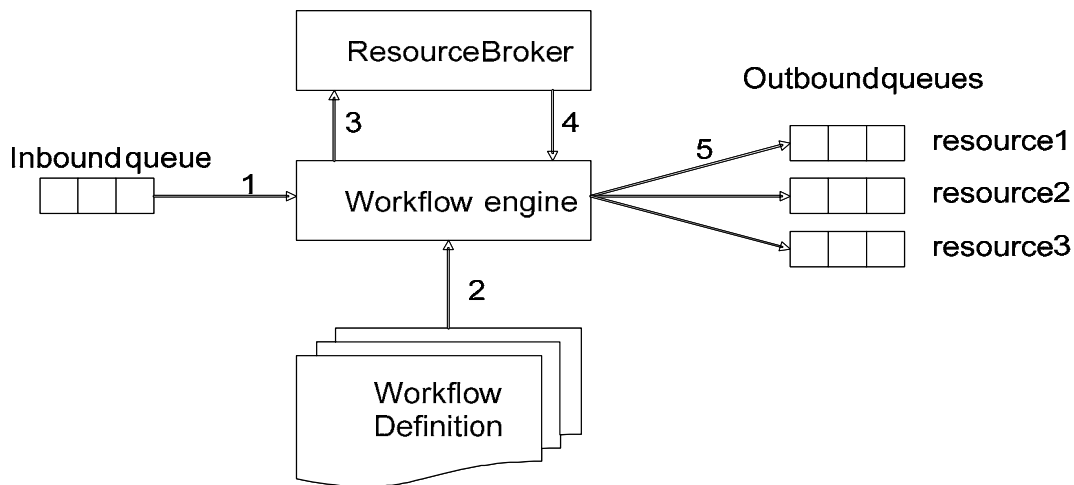


Figure 2.6: A Typical Workflow Engine [Alo04]

Administration & Monitoring Tools Interface: This is one of the most important units of a WfMS. It basically defines monitoring and fault handling functions. The monitoring tools support following functions [WFMC].

- Track and monitor individual work requests
- Review resource productivity and work volume analysis
- Quickly search for and identify a work request
- Provide feedback on performance issues
- Get information about the bottlenecks in the process
- Analysis to implement changes to the workflow process

Failure handling is very critical and a proper fault handling mechanism is a must for useful workflow management systems. There are several techniques such as forward recovery, backward recovery, and exception handling to handle errors in a workflow [WFMC].

3. Workflow Interoperability Interface: This interface defines protocols and technologies to inter-operate among workflow activities. Workflow activities give rise to network and systems heterogeneity. The interoperability interface defines a set of interoperable protocols [WFMC]. Section 2.2 discusses issues of this type of interoperability in detail and reason out how web services solve such interoperability problems.

4. Workflow Client Application Interface: This refers to the definition of APIs for client applications to request services from the workflow engine to control the progression of processes, activities and work-items.

5. Invoked Application Interface: This is the standard interface definition of APIs that allow the workflow engine to invoke a variety of applications, through common agent software.

In this section, we have discussed the essential details about what workflows are and what is involved in defining and executing workflows. As we have mentioned earlier, interoperability among workflow entities is one of the pressing issues. Section 2.2 discusses the interoperability problem in detail and illustrates how web services solve the interoperability problem.

2.2 Information Systems

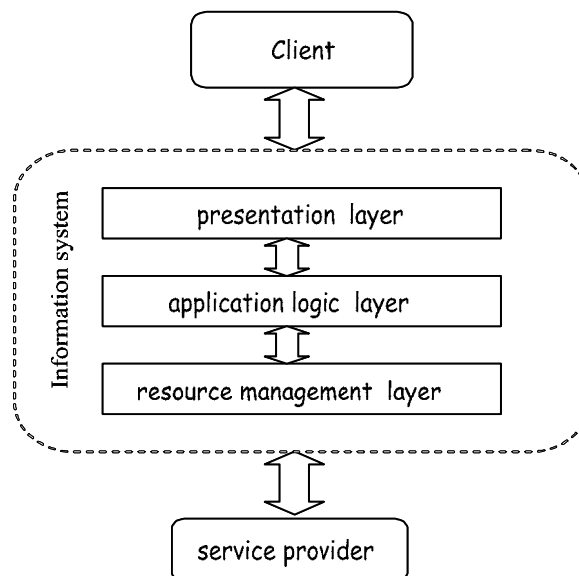


Figure 2.7: A Typical Information System [Alo04]

Figure 2.7 illustrates the architectural organization of a typical information system. In an automated workflow, these information systems interact among each other to enact workflow requirements. Thus, it is important to have a better understanding of the technology behind information systems. Any information system has three distinct layers, namely requesters, an information management system, and service providers. First, we will focus on the implementation of the information system.

At an abstract level, information systems are designed around three layers: presentation layer, application layer, and resource management layer [Bra03]. The way these layers are arranged between service provider and the service requester determines if it is 1-tier, 2-tier, 3-tier, or n-tier. In 1-tier systems, all three layers have been implemented by a single system and hosted in a single machine. Service requesters interact with the interface of the system to get services rendered. Such systems are similar to the early MainFrame systems. Clients of such systems act as dummy terminals.

2-tier systems are classical client server systems where both clients and the information systems execute stubs related to the service. Java RMI, CORBA and DCOM are classic examples of such systems. The 3-tier and multi-tier systems use one or more middleware components to render services to clients. Figure 2.7 illustrates different architectures of information systems.

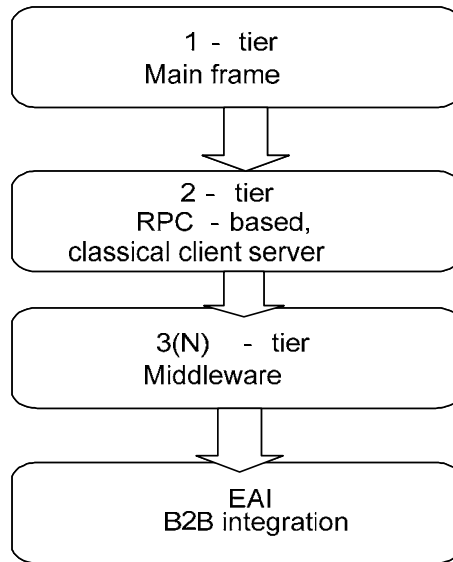


Figure 2.8: B2B Integration

Currently, many information systems are multi-tier systems. In the world of workflows, underlying implementations of information systems are important because, it is required to integrate several such systems together to form sophisticated workflows. These systems are maintained by different organization having various propriety requirements.

Inter-organizational integration is handled mainly in the following three layers.

- Presentation layer.
- Middleware layer.
- Application layer.

However, the development of inter-organizational workflows is handicapped due to several reasons: a) organizations are reluctant to expose their application logic, b) There is heterogeneity of application logic and technology used. Figure 2.9 further illustrates this interoperability problem.

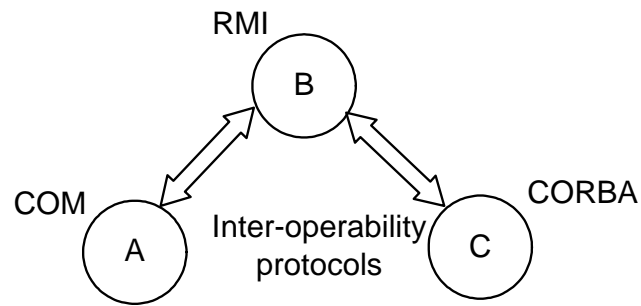


Figure 2.9: The Interoperability Problem

Suppose a workflow has a requirement to integrate three systems A, B, and C. Also, suppose these systems have been developed using COM, RMI, and CORBA. Applications developed using these technologies do not communicate with each other directly. Different inter-operability protocols are needed and this solution is not scalable.

From Conventional Information Systems to Web services: In order to solve the interoperability problem, service providers need to have a universally accepted interface for their services. This has led to the idea of “*service oriented computing*”. Service providers publish the interface of their services so that other entities can find and use them. Vendors use universally accepted Internet protocols to exchange data and service invocation. Since these information services are distributed across the Internet, they are called *web services* (Figure 2.10).

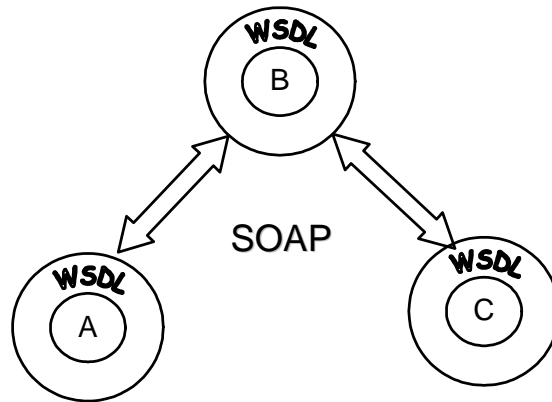


Figure 2.10: Web Services: A Uniform Interface and a Common Communication Protocol

2.3 Web Services

Web service is defined as “a software application identified by a URI, whose interface and binding are capable of being *defined*, *described*, and *discovered* as XML artifacts. Web service supports direct *interactions with other software agents* using XML based messages and exchanged via internet-based protocols” [Aal02] (W3C).

Description (WSDL)
Discovery (UDDI)
Interaction (SOAP)
Coordination/compo sition (BPEL)

Figure 2.11: Web Service Definition

Web services hide the system and network heterogeneities using uniform interface (WSDL) and a common communication protocol (SOAP). With these features, web services bring the loosely coupledness to information systems. They are loosely coupled because the service developments and the application developments are totally orthogonal and service requesters can dynamically query available services and bind them to the application at runtime. This means that web services need a repository to register them so that requesters can query and find them at runtime. Figure 2.12 illustrates the current web services infrastructure. Service developers publish their services in the UDDI (or other local) directory and applications (application developers) look up the directory for required services.

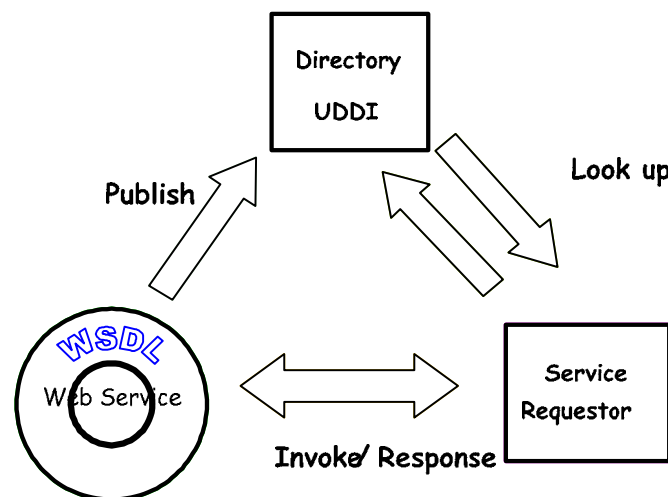


Figure 2.12: Current Web Services Infrastructure

Web service based applications are developed by composing different web services together. These composite applications are typically long running. Figure 2.13 shows the current state of the art in developing composite web service applications. The composite

web service handles communication as well as the application logic. Majority of web service based applications are workflows so that they have to model and enforce most of the workflow requirements described in the previous section [Aal03a]. New protocols and standards are required to develop these applications efficiently. There is a plethora of overlapping and competing standards for web service coordination. However, the technology itself is relatively new and is in the development stage.

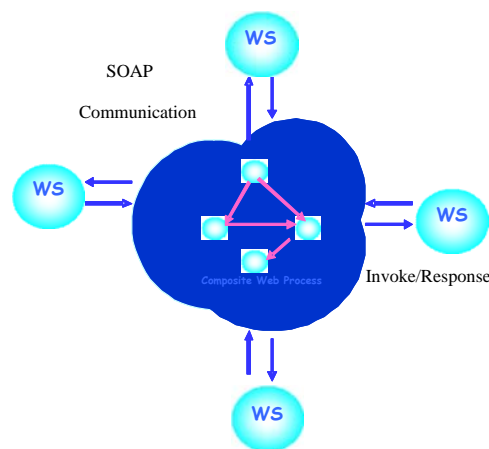


Figure 2.13: Web Service Composition

Figure 2.14 shows the taxonomy of current web service protocols and languages. In this dissertation we focus on web service composition and middleware support for web service composition.

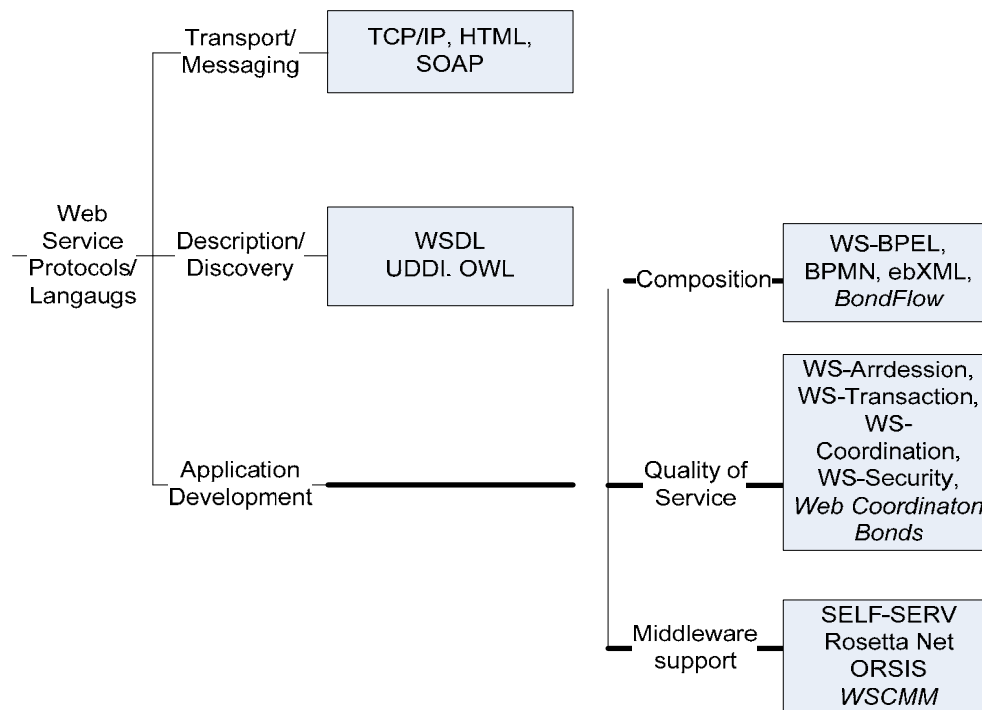


Figure 2.14: Taxonomy of Web Service Standards

In chapters 3 and 4, we present web coordination bonds as a set of primitives for web services coordination and dependency modeling. Chapters 5 and 6 present our WSCMM for distributed workflow coordination web services. Finally, we present the BondFlow system as a workflow composition and deployment engine.

2.4 Merging Web Service and Workflows

While web service technology is proliferating, workflow management systems are moving forward in parallel. Figure 2.15 shows the chronology of workflow languages and systems. Originally this figure has been published in [Mue05]. We have modified it to add current developments. It is important to note that XML is becoming the common technology for workflow specifications. Current standard XPDL has been evolved with many ideas from web service standard organizations such as OASIS and W3C. These developments clearly indicate that future workflow applications depend on the Internet and web service technologies.

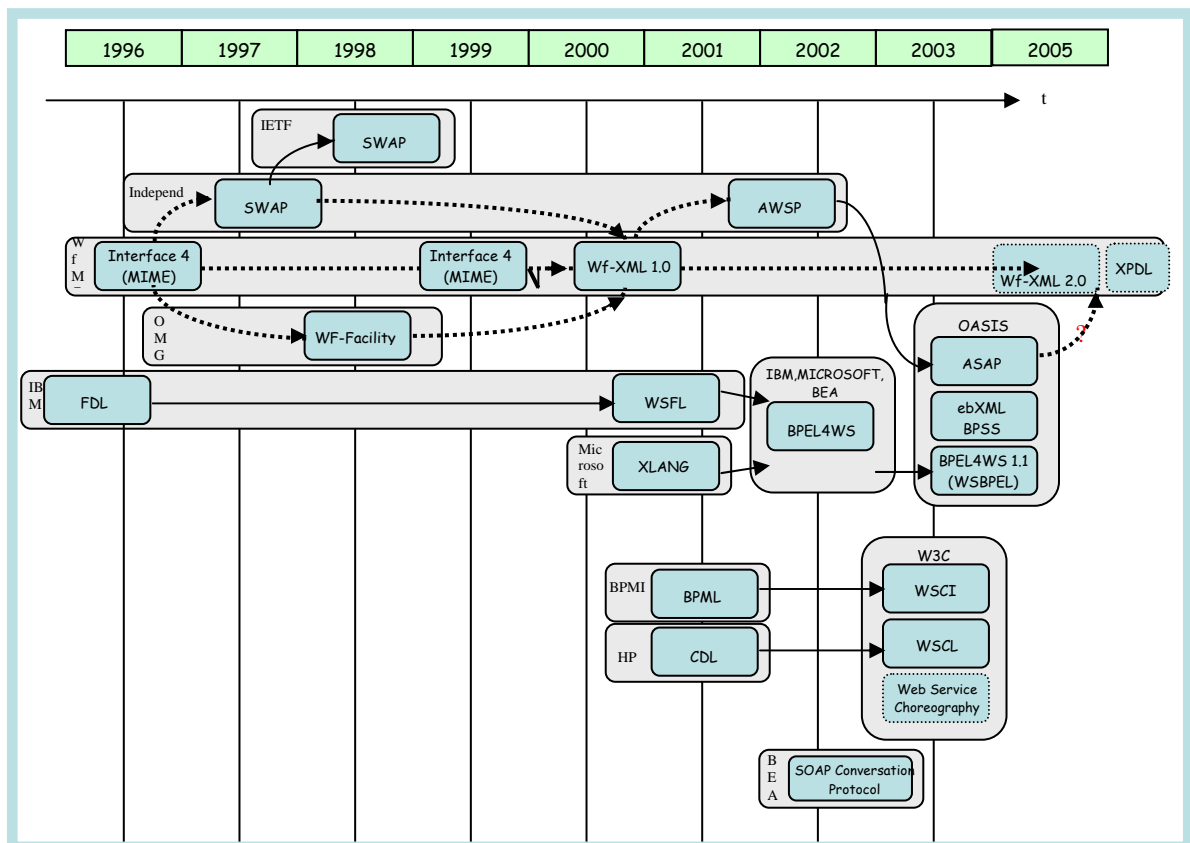


Figure 2.15: Workflow Language and Protocol Chronology [Mue05]

As we have mentioned, contributions of this dissertation work are three fold: a) a set of coordination primitives for web service workflow dependency modeling, b) a middleware framework for distributed workflow coordination, and c) a system to configure and execute workflows. Each subsequent chapter discusses more specific technologies and related work in detail.

CHAPTER 3

WEB COORDINATION BONDS

In this chapter we introduce web coordination bonds (also alternatively called “web bonds” for short, or “coordination links” to generalize to web and non-web entities) as a capable set of primitives for web service coordination. The idea of web coordination bonds originated from our study of how to setup a meeting using online calendars of schedules of people with automatic negotiation among calendars in case of individual cancellations. The result was the artifact called coordination links to establish and enforce dependencies among collaborating entities [Pra03a, Pra03b, Pra04a].

First we present the idea of web bonds [Pra04b]. Then, we formally define a network of over objects and prove that web bonds are at least as powerful as the Petri nets extended with inhibitor arcs. Web bonds can establish (model) and enforce (deploy and execute) dependencies of various kinds [Pra05]. Next, we demonstrate this for producer-consumer relationships and shared-resource relationships. These two kinds of relationships have been shown to yield the fundamental categories of dependencies [Mal94]. A detailed meeting setup example is also presented to further illustrate the resource-sharing paradigm. Finally, we survey the relevant literature, and compare and contrast with web coordination bonds.

3.1 Introduction

Web coordination bonds are analogous to the chemical bonds in chemical compounds, which are too simple yet extremely powerful to enable all sorts of basic and complex

chemical compounds to exist naturally and to be manufactured artificially. Different atoms expose sites with certain number of either excess or shortage of electrons. For example, oxygen atom has two negatively charged sites, and hydrogen has a deficit of one electron, giving it a positively charged site. To form a water molecule, therefore, two hydrogen atoms bond with an oxygen atom - each bond is just a sharing of an electron between a donor and a recipient site. The web services are simple or composite server objects situated on the web with well-defined interfaces and are the “*web atoms*.” Molecules are, therefore, analogous to all collaborating processes involving individual web service components. The list of such “*web molecules*” spans transient to long-running collaborative processes, transactions, client-server and p2p distributed applications, workflows as well as virtual organizations. Taking the analogy further, the challenge is to (i) to define the analogous “bonding sites” or simple “*web hooks*” in the web service interface needed to mesh multiple web entities together, and (ii) to develop the analogous concept of a few simple yet powerful types of “*web bonds*” which would be the coordination threads to bind and produce the “*Web molecules*” out of multiple “*web atoms*.” These “*web bond*” primitives should allow rapid modeling and deployment of collaborative applications of all kinds and complexities (Fig. 3.1).

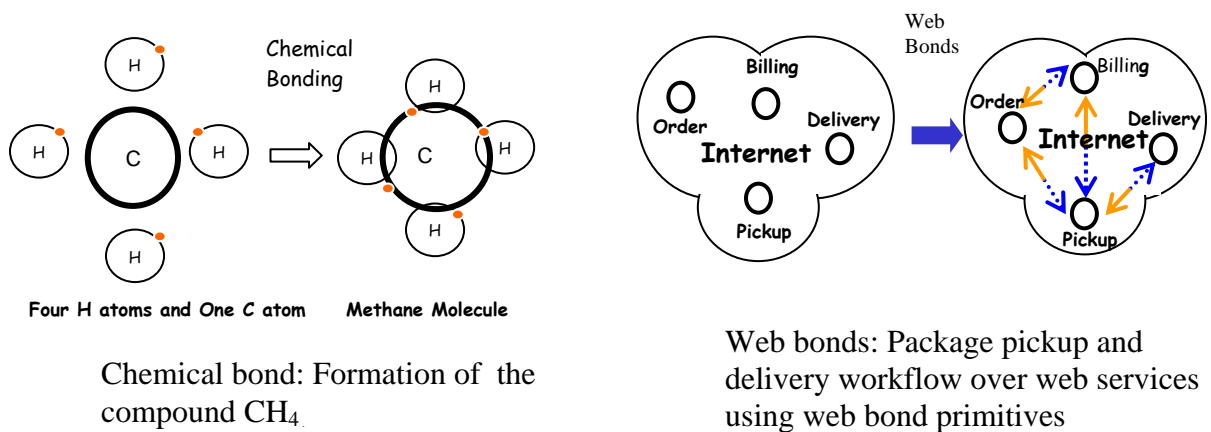


Figure 3.1: Analogy between Chemical Bonds and Web Bonds

Currently, the hooks exposed by the web services are the basic methods published and the bonds available are no more capable than the one-time invocations of those methods by a client web entity.

3.2 Web Coordination Bond Concepts

Web bonds enable applications to create contracts between entities and enforce interdependencies and constraints, and carry out atomic transactions spanning over a group of web entities/processes. We define two types of web bonds: *subscription bonds* and *negotiation bonds*. The subscription bond allows automatic flow of information from a source entity to other entities that subscribe to it. This can be employed for synchronization as well as more complex changes, needing data, control, or event flows. Negotiation bonds enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction.

A web bond is specified by its type (subscription/negotiation), references to one or more web entities, triggers associated with each reference (event-condition-action rules) [Pat99], a priority, a constraint (AND, OR, XOR), and a bond creation expiry time [Pra04b, Pra04a]. Let an entity *A* be bonded to entities *B* and *C*, which may in turn be bonded to other entities. A change in *A* may trigger changes in *B* and *C*, or *A* can change only if *B* and *C* can be successfully changed. In the following, the phrase "Change *X*" is employed to refer to an action on *X* (action usually is a particular method invocation on web service *X* with a specified set of parameters); "Mark *X*" refers to an attempted change, which triggers any associated bond without an actual change on *X*.

Subscription-and Bond: Mark A; If successful then Change A and Try: Change B, Change C.

A ``try" may not succeed. Similarly, subscription-or and subscription-xor bonds can be defined.

Negotiation-and Bond: Change A only if B and C can be successfully changed.

(Implements atomic transaction with "and" logic)

Semantics (shown for the illustration, but may have alternative implementations):

Mark A for change and Lock A

If successful

Mark B and C for change and Lock B and C

If successful to lock both B and C

Change A

Change B and C

Unlock B and C

Unlock A

Note that locks are only for the explanation of the semantics. A reservation/locking mechanism to implement this usually will have an expiry time to obviate deadlocks. In a database web service, this would usually indicate a "ready to commit" stage.

Negotiation-or Bond: Change A only if at least one of B and C can be successfully changed. (Implements atomic transaction with "or" logic and can be extended to at least k out of n).

Semantics:

Mark A for change and Lock A

Mark B and C for change; Obtain locks on those entities that can be successfully changed.

If at least one lock is obtained

Then *Change A; Change the locked entities.*

Unlock entities

Negotiation-xor Bond: Change A only if exactly one of B and C can be successfully changed. (implements atomic transaction with "xor" logic and can be extended to *exactly k out of n*).

Semantics:

Mark A for change and Lock A

Mark B and C for change. Obtain locks on those entities that can be successfully changed.

If exactly one lock is obtained

Then *Change A ; Change the locked entities.*

Unlock entities

A negotiation bond from A to B has two interpretations: pre-execution and post-execution. In case of pre-execution, in order to start the activity A , B needs to complete its execution. In case of post-execution, in order to start the activity A , A needs to make sure that B can be completed afterwards. Both pre- and post-execution interpretations of negotiation bonds enforce atomicity. In the rest of the paper, unless specified, we have employed the pre-execution type of negotiation bonds implicitly.

Web bonds can be *tentative* or *confirmed*. Confirmed bonds receive messages and trigger appropriate actions. Tentative bonds are in waiting state to become confirmed. They are in the waiting state due to reasons such as less priority and inadequate resources. Usefulness of tentative bonds can be explained using the following meeting example. Suppose an attendee cannot commit for a meeting at the time meeting is scheduled, but the initiator still wants to schedule a tentative meeting, pending changes in the schedule of the attendee at a later time. If this attendee is a “must” attendee, then there is a tentative bond created back to the initiator. Typically, the reason that an attendee cannot commit is because of a prior commitment, and hence a non-tentative confirmed negotiation bond. Many such tentative bonds may go out from an attendee,

and therefore, these tentative bonds are in a priority queue of waiting list. If and when the confirmed bond is destroyed, the highest priority tentative bond in the waiting list is converted to a confirmed bond, and the associated trigger is activated. This trigger could allow the initiator of the meeting to resolve the conflicts for this meeting and declare it committed.

3.2.1 Notations for Web Bonds

A subscription bond from A to B is denoted as a *dashed directed arrow* from A to B . A negotiation bond from A to B is denoted as a *solid directed arrow* from A to B . A negotiation-and bond from A to B and C is denoted by two solid arrows, one each to B and C , with a "*" in between the arrows. Similarly, a negotiation-or bond from A to B and C is denoted by two solid arrows, one each to B and C , with a "+" in between the arrows. A negotiation-xor bond from A to B and C is denoted by two solid arrows, one each to B and C , with a "^" in between the arrows. A tentative bond, which is a negotiation bond in a waiting list, is shown as a solid arrow with cuts.

As shown in Figure 3.2, if there is a subscription bond from activity A to activity B , it implies that once A completes its execution (or, completes some functionality indicated by the subscription bond), B will be notified with suitable control and data as specified by the subscription bond.

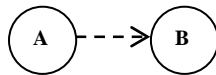


Figure 3.2: Subscription bond

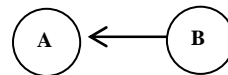


Figure 3.3 Negotiation bond

Negotiation bonds enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction. If there is a negotiation bond from activity B to activity A (Figure 3.3), it has two interpretations: pre execution and post execution. In case of pre-execution, in order to start activity B , A needs to complete its execution. In

case of post-execution, in order to start activity B , B needs to make sure that A can be completed afterwards. In this dissertation, we have primarily employed the pre-execution type of negotiation bonds implicitly.

Methods of activities can be bonded using both types of bonds simultaneously. This special case is denoted as subscription-negotiation bond pair (Figure 3.4).

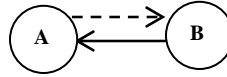


Figure 3.4 Subscription-negotiation bond pair

Subscription-negotiation bond pair enforces the following condition. In order to execute B , the activity A must be completed, and in addition, A can inform B of its execution by sending control and/or data to B .

3.3 Evaluating Capabilities of Web Coordination Bonds

Efficient and effective distributed coordination require solid, unambiguous set of primitives with sufficient expressive capabilities to bond (hook) autonomous constituent parties together to form a coherent unit. Expressive power of a language has been generally linked to its suitability. In our context, the primitives which make up such a language should have enough *expressive power* to model complex processes, *clearly defined semantics* [Bus03] to avoid ambiguity, and enough *analytical power* to learn about and verify the correctness [Tho03]. To illustrate expressive and modeling power, consider a comparison between C++ vs. Java when we need to program a GUI interface. In terms of modeling capabilities both languages are Turing complete. However, one can write such a program easily using Java's swing package that may require much more

effort in C++. Thus, Java turns out to be more expressive in this case. The difference between modeling power and expressiveness is that the former indicates the ability to design or model coordination (interaction) patterns whereas the latter denotes how efficiently and easily such patterns can be modeled. In other words, modeling power can be regarded as the theoretical limit, whereas expressive power can be regarded as the practical limit. Thus, it is important to access both the modeling power and the expressive power to evaluate capabilities of web bonds.

In this chapter, we prove that web bonds can model extended Petri nets, and thus, are fundamentally capable primitives [Woh02]. In the literature, authors have generally agreed on some standard workflow control flow and distributed communication patterns. It is our intent to prove in the next chapter that web bonds can indeed model such patterns.

3.4 Modeling Power of Web Bonds

In this section, we prove the modeling power of web coordination bonds in terms of Petri nets. Petri nets have been employed as a benchmark to access the capabilities of object oriented programming, showing that object-oriented features can be mapped directly onto behaviorally equivalent colored Petri nets [Jen87, Lak94, Lak95].

We establish here that web bonds have the modeling power of extended Petri nets. This is important because extended Petri nets are the most powerful among different Petri net models and is equivalent to the Turing machine [Age74, Mur89]. We prove this by simulating the transitions that an extended Petri net can carry out by employing a network of web bonds over stateful objects. An extended Petri net has places, transitions, and two kinds of arcs, normal arcs and inhibitor arcs that link places to corresponding transitions.

A transition can fire if and only if each input place associated with normal arcs has a token that can be consumed by the transition and each input place associated with inhibitor arcs has no token. Firing of a transition results in placing a token in each output place. We model each place as an object having methods to consume a token, add a token, and check if it has zero tokens. A transition is modeled as an object which “fires” if and only if the input and output objects satisfy the condition for firing the transition. It employs a suitable network of negotiation bonds to enforce this dependency.

Before going into details of the proof, first we formally define the Petri nets with inhibitor arcs as the Extended Petri Nets (EPN) and also give a formal definition of a network of web bonds.

Extended Petri Net (EPN): A EPN is defined as a 4-tuple (P, T, A, f) [Age73], where

$T = \{t_1, t_2, \dots, t_n\}$ is a finite set of transitions,

$P = \{p_1, p_2, \dots, p_m\}$ is a finite set of places,

$A = \{T \times P\} \cup \{P \times T\}$ is a finite set of directed arcs such that

$$(p_i, t_j) \in A \Rightarrow (t_j, p_i) \notin A,$$

$$(t_j, p_i) \in A \Rightarrow (p_i, t_j) \notin A, \text{ and}$$

$f: A \rightarrow \{\text{True}, \text{False}\}$ indicates if an arc is a normal arc or an inhibitor arc.

Two sets I_i', I_i'' are defined as follows for a given transition t_i :

$I_i' = \{j \mid (p_j, t_i) \in A \text{ and } f(p_j, t_i) = \text{True}\}$, is the set of indices of the places which have normal arcs to transition t_i .

$I_i'' = \{j \mid (p_j, t_i) \in A \text{ and } f(p_j, t_i) = \text{False}\}$, is the set of indices of the places which have inhibitor arcs to transition t_i .

Transition firing rule: A transition t_i is enabled if its input places have at least one token each except for those places which have inhibitor arcs to t_i , which must have zero tokens each. That is, for each arc $(p_j, t_i) \in A$, $p_j > 0$ for all $j \in I_i'$ and $p_j = 0$ for all $j \in I_i''$. An

enabled transition can fire. When t_i fires, it atomically i) deletes a token from each input place p_j for all $j \in I_i'$, and ii) puts a token in each output place p_j where $(t_i, p_j) \in A$.

A Network of Web Bonds (WB) is defined as a 2-tuple (O, B) , where

$O = \{o_1, o_2, \dots, o_n\}$ is a finite set of objects, and

$B = \{b_1, b_2, \dots, b_m\}$ is a finite set of bonds.

An *object* o_i is a 2-tuple (M, V) , where $M = \{m_1, m_2, \dots, m_{|M|}\}$ is a finite set of methods available at o_i and $V = \{v_1, v_2, \dots, v_{|V|}\}$ is a finite set of data variables [Lak94]. We use the notation $o_i.m_j(\text{param}_k)$ to denote the method m_j of object o_i with parameter set param_k .

A *bond* b_i is a 3-tuple (s, D, Type) , where

$s = o_i.m_j(\text{param}_k)$ is the source method,

$D = \text{set of one or more destination methods } o_i'.m_j'(\text{param}_k'), \text{ and}$

$\text{Type} \in \{\text{Subscription, Negotiation}\}$.

Subscription bond: A subscription bond from method m_j with parameter set param_k of object o_i to method m_i' with parameter set param_k' of object o_i' is defined as follows:

if $o_i.m_j(\text{param}_k)$ is executed then invoke $o_i'.m_i'(\text{param}_k')$.

Negotiation-and bond: A negotiation-and bond from method m_j with parameter set param_k of object o_i to each of $o_i'.m_j'(\text{param}_k') \in D$ is defined as follows:

execute $o_i.m_j(\text{param}_k)$ only if all $o_i'.m_j'(\text{param}_k') \in D$ can be executed.

Theorem: *Web bonds have the modeling power of Extended Petri Nets as defined above.*

Proof: To prove this we map a generic EPN to a network of web bonds as follows

(Figure 3.5 and Figure 3.6).

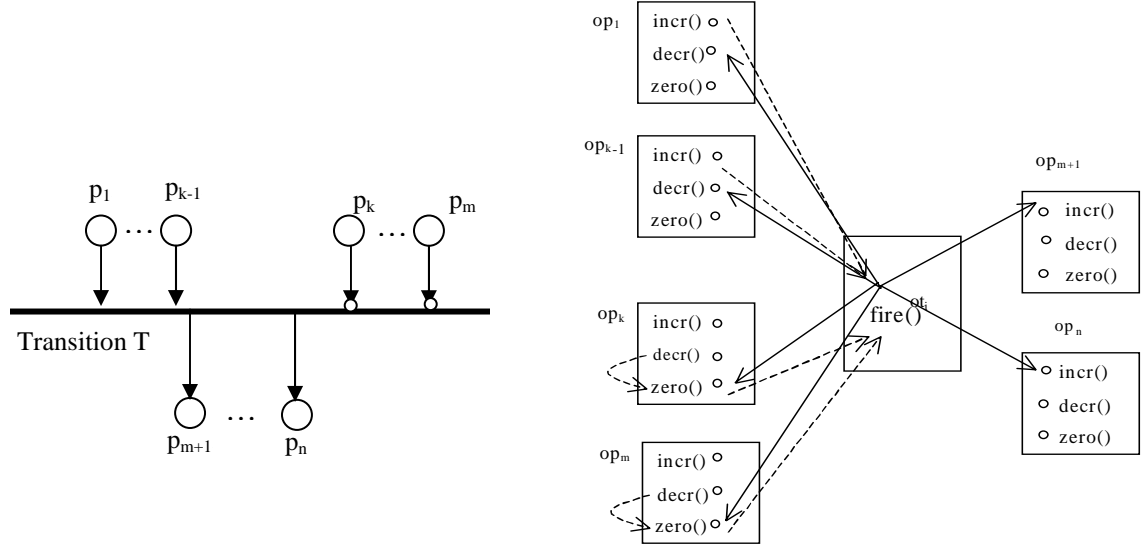


Figure 3.5: Petri Net with inhibitor arcs (EPN) Figure 3.6: Simulating EPN using web bonds

We define a network of web bonds $WB(O, B)$ corresponding to a $EPN(T, P, A, f)$. Set O is a collection of two types of objects, corresponding to the places and the transitions of EPN, defined as follows (Figure 3.6).

$O = P' \cup T'$ such that $P' = \{op_j \mid p_j \in P\}$, $T' = \{ot_i \mid t_i \in T\}$.

Each $op_j \in P'$ and $ot_i \in T'$ has the following methods and data variables.

$op_j = (\{\text{increment}(), \text{decrement}(), \text{zero}()\}, \{\text{int num_tokens}\})$, and

$ot_i = (\{\text{fire}()\}, \{\})$, where,

increment():

num_tokens ++;

return true;

zero():

if (num_tokens == 0)

return true;

else

return false;

decrement():

if (num_tokens > 0)

{ num_tokens --;

return true;

}

else return false;

fire():

return true;

For each t_i , its set of incident arcs is mapped to a negotiation-and bond

$b_i = (ot_i.\text{fire}(), D_i, \text{negotiation-and})$ in B

with set of destination methods D_i defined as follows:

$D_i = \{op_j.\text{decrement}() \mid j \in I'_i\} \cup \{op_j.\text{zero}() \mid j \in I''_i\} \cup \{op_j.\text{increment}() \mid (t_i, p_j) \in$

$A\}$.

In addition to these negotiation bonds among the objects in O to carry out the transition firing, there are three sets of subscription bonds in B for event flows whenever tokens change:

$\{(op_j.\text{increment}(), ot_i.\text{fire}(), \text{subscription}) \mid j \in I'_i\} \cup$

$\{(op_j.\text{zero}(), ot_i.\text{fire}(), \text{subscription}) \mid j \in I''_i\} \cup$

$\{(op_j.\text{decrement}(), op_i.\text{zero}(), \text{subscription}) \mid j \in I'''_i\}$.

The first and the second set of subscription bonds, respectively, check for firing transition t_i after an additional token is received in an input place and an inhibitor input place reaches zero tokens. Third set invokes zero token checking in inhibitor places after each decrement.

We define three sets J'_i , J''_i , and J'''_i in WB as follows.

- i) $J_i' = I_i'$,
- ii) $J_i'' = I_i''$, and
- iii) $J_i''' = \{j \mid (t_i, p_j) \in A\}$.

With that we prove that when transition $t_i \in T$ of EPN fires there is a corresponding execution of `fire()` method of object $ot_i \in T'$ of WB, and vice-versa.

Part I: For each firing of transition $t_i \in T$ of EPN there is a corresponding execution of `fire()` method of object $ot_i \in T'$ of WB.

When transition $t_i \in T$ of EPN fires, it atomically deletes a token from each input place p_j for all $j \in I_i$, puts a token in each output place p_j where $(t_i, p_j) \in A$ and makes sure that $p_j = 0$ for all $j \in I_i''$. Correspondingly, because of the negotiation-and bond b_i according to the above mapping, when `ot_i.fire()` method in WB is executed successfully, then, atomically, all `op_j`'s for all $j \in J_i'$ execute `op_j.decrement()` method, all `op_j`'s for all $j \in J_i'''$ execute `op_j.increment()` method, and all `op_j`'s for all $j \in J_i''$ execute `op_j.zero()` method successfully.

Part II: For each execution of `fire()` method of object $ot_i \in T'$ of WB, there is a corresponding firing of transition $t_i \in T$ of EPN.

According to the mapping, when `ot_i.fire()` method of object $ot_i \in T'$ of WB is executed, it atomically executes `op_j.decrement()` method in `op`'s for all $j \in J_i'$, `op_j.increment()` methods in `op_j`'s for all $j \in J_i'''$, and `op_j.zero()` method for all $j \in J_i''$. Correspondingly, the transition $t_i \in T$ of EPN is enabled and its firing atomically deletes a token from each input place p_j for all $j \in I_i'$, puts a token in each output place p_j where $(t_i, p_j) \in A$, and makes sure that $p_j = 0$ for all $j \in I_i''$.

Corollary: Web bonds can simulate the operation of a Turing machine.

Proof: Extended Petri nets as defined above and Turing machines are equivalent [Age74]. We have shown that the extended Petri net can be simulated using a network of

web bonds. Therefore, web bonds can simulate the operations of Turing machines. Thus, web bonds are fundamentally sound in terms of their modeling power.

3.5 Modeling Various Dependency Scenarios Using Web Coordination Bonds

Expressive capabilities of web bonds can be illustrated through typical scenarios of dependencies. In [Mal94], authors have identified common dependencies between activities such as producer/consumer and shared resources. In this section, we illustrate how such dependencies can be modeled using web coordination bonds.

3.5.1 Producer-Consumer Dependencies

Figure 3.7 shows how a classic relationship of a producer and consumer web process can be modeled using two negotiation bonds. The “*Place_Order*” method at a consumer process needs to ensure that the producer has enough inventories such that the corresponding “*Accept_Order*” method will get executed successfully.

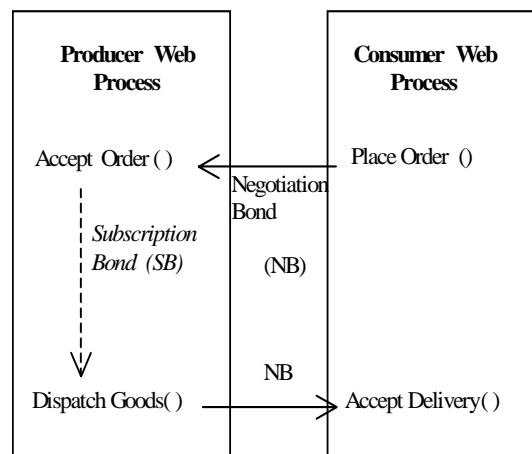


Figure 3.7: Coordinating Producer-Consumer web Processes

Before guaranteeing this, the “*Accept_Order*” probably will check the current and projected inventory. A negotiation bond is created from consumer web process to producer web process. This is the basic situation for deploying a negotiation bond. Once order has been placed by the consumer and accepted by the producer, a subscription bond serves notice to “*Dispatch_Goods*” method. Note that the web bonds are useful within a web process as well. Again before “*Dispatch_Goods*” executes, it needs to ensure that consumer’s “*Accept_Delivery*” method can be completed successfully (ensuring that enough space is available, for example).

Figure 3.8 illustrates how multiple producer scenarios can be easily integrated with a consumer. “*Call_for_Bids (I, C)*” is executed announcing solicitation of bids (at least I , an installment, but no more than C , the capacity). At all the producers, which have subscribed to this method at the consumer, their “*Place_Bid*” method is activated. Those producers, who are able and willing to place bids successfully, activate the “*Select_Bid*” method of the consumer. The subscription bonds carry out these two steps, as no negotiation is needed. Once a successful bid of a Producer PB_{IB} has been chosen, the subscription bond from “*Select_Bid ()*” is triggered, which activates the “*Place_Order*” method at the consumer, and the scenario as in the previous paragraph gets carried out.

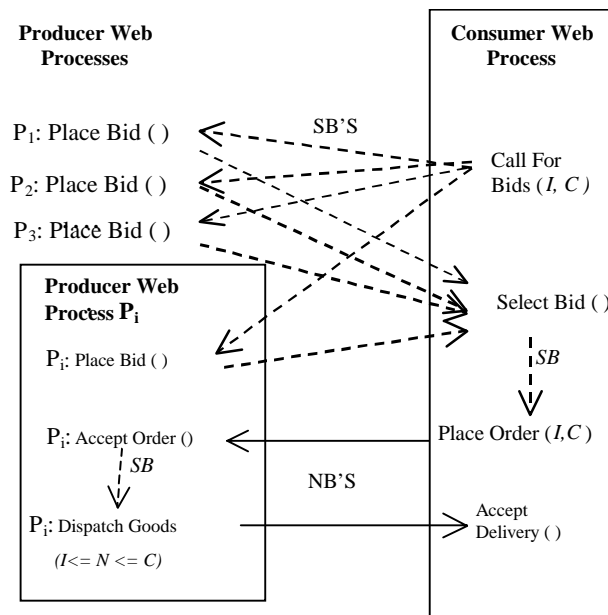


Figure 3.8: Coordinating multiple producers with a consumer Web process

3.5.2 Shared Resource Dependencies

Modeling dependencies between competing entities for a shared resource is natural to web bonds.

Resource Allocation

Figure 3.9 shows the bonds needed for two processes A and B to compete for a shared resource process. The “*Acquire*” method of competing processes have a negotiation bond to the “*Allocate*” method of the shared resource web process; unless “*Allocate*” can be guaranteed, “*Acquire*” can not succeed. Note that “*Allocate*” will guarantee reservation/lock to only one requesting process, say A, by creating a negotiation bond back to A, while wait-listing B’s request using a tentative bond back to B (Figure 3.9b). Subsequently A executes its “*Release*” thereby de-allocating its reservation and thus deleting the negotiation bond that was created from the shared resource to A. This will

change the tentative bond to B into a confirmed bond, triggering a round of negotiation with “Acquire” process of B (Figure 3.9c).

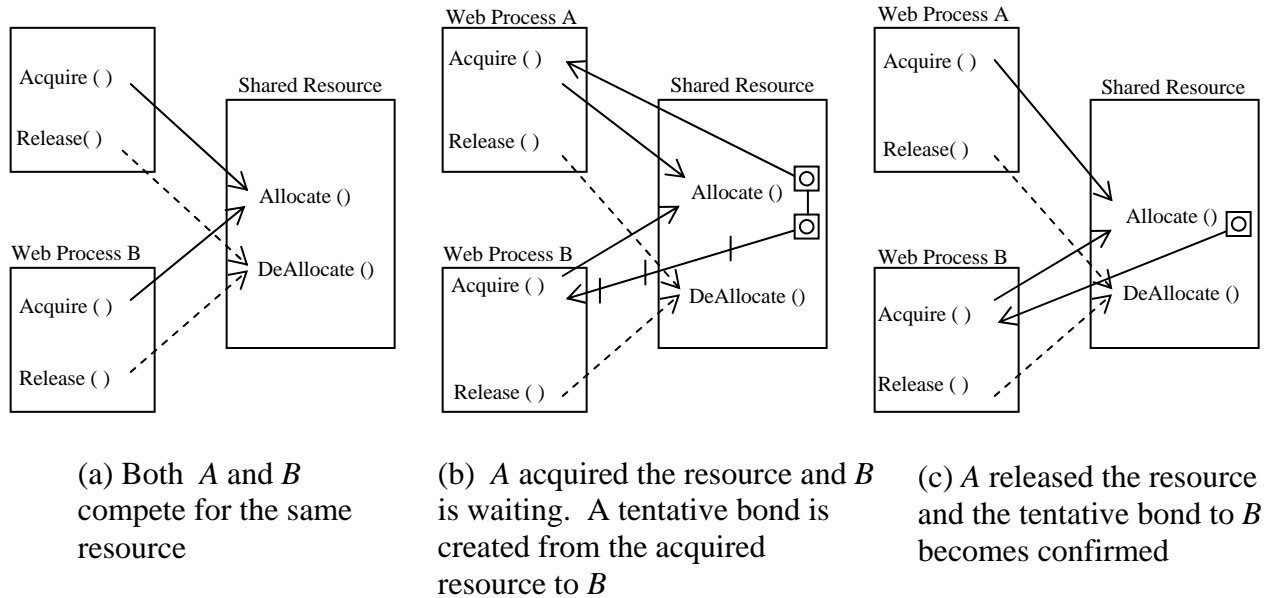


Figure 3.9: Modeling Resource Sharing among Competing Web Processes

3.5.3 An Application Scenario: Shared Calendars of Meeting Example

The potential of web-bond-like primitives and their utility in modeling and enforcing contracts among competing web services can be further illustrated by a calendar of meeting example. For this application, we demonstrate here how an empty time slot is found, how a meeting is setup (tentative and confirmed), and how voluntary and involuntary changes are automatically handled. A simple scenario is as follows: *A* wants to call a meeting between times t_1 and t_2 involving *B*, *C*, *D* and himself. The first step is to invoke *Get_Available_Times()* method to find the empty slots in everybody's calendar. *A* then reserves the desired empty slot by calling *Setup_Meeting(t_1, t_2)* method. This causes a series of steps. A negotiation-and bond is created from *A*'s slot (t_1, t_2) to each of

the calendar tables ($A.Setup_Meeting(t_1, t_2)$, $\{A.Reserve_Slot(t_1, t_2), B.Reserve_Slot(t_1, t_2), C.Reserve_Slot(t_1, t_2), D.Reserve_Slot(t_1, t_2)\}$, negotiation-and) (Figure 3.10).

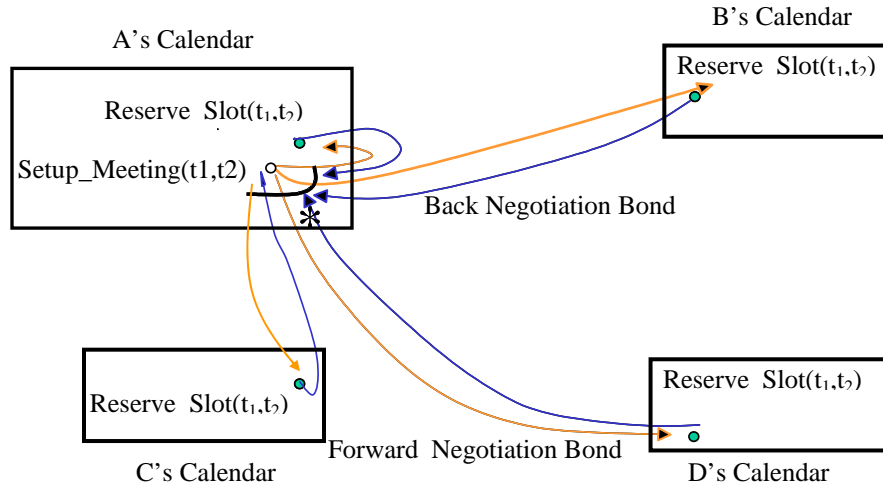


Figure 3.10: Scheduled Meeting

If slot can be reserved, then each corresponding slot at A , B , C and D create a negotiation bond back to A 's slot. That is ($A.Reserve_Slot(t_1, t_2)$, $\{X.Setup_Meeting(t_1, t_2) \mid X \in \{A, B, C, D\}\}$, negotiation-and) are created.

Else, for those folks who could not be reserved, a tentative bond back to A is queued up at the corresponding slots to be triggered whenever the status of the slot changes. The forward negotiation-and bond to A , B , C and D are left in place. Back subscription bonds to A from others are created to inform A of subsequent changes in the other participants and to help A decide to cancel this tentative meeting or try another time slot.

Assume that C could not be reserved. Thus, C has a tentative bond back to A , and others have subscription bonds to A (Figure 3.11). Whenever C becomes available (i.e., $Release_Slot(t_1, t_2)$ method is invoked), if the tentative bond back to A is of highest priority, it will get triggered, informing A of C 's availability. This triggers the negotiation-and bond from A 's $Setup_Meeting(t_1, t_2)$ to $Reserve_Slot(t_1, t_2)$ of A , B , C and

D , resulting in another round of negotiation. If all succeed, then corresponding slots are reserved, and the target slots at A , B , C and D create negotiation bonds back to A 's $Setup_Meeting(t_1, t_2)$ method (Figure 3.10). Thus, a tentative meeting has been converted to committed state.

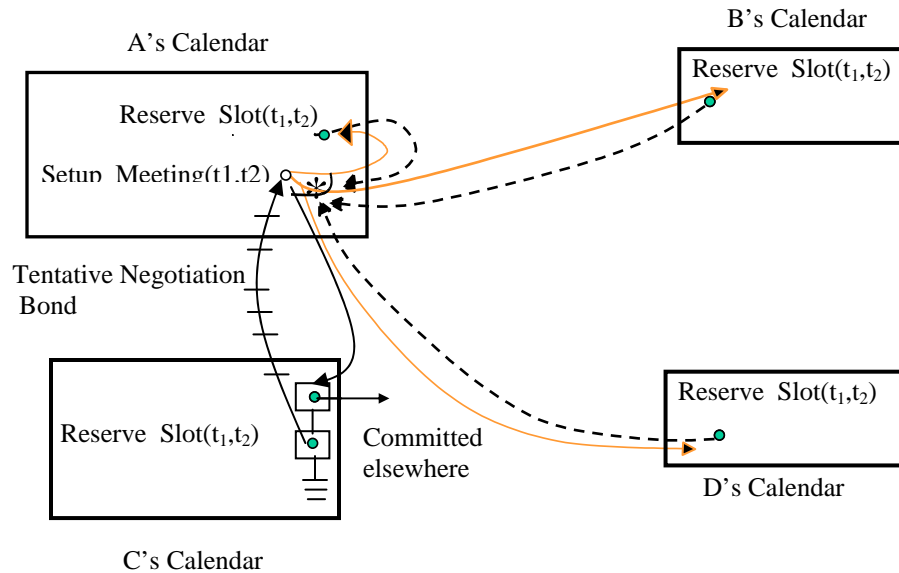


Figure 3.11: A Tentative Meeting

Now, suppose D wants to change the schedule for this meeting. This would trigger its bond to A , triggering the forward negotiation-and bond from A to A , B , C and D . If all succeed, then a new duration is reserved at each calendar with all forward and back bonds established. If not all can agree, then D would be unable to change the schedule of the meeting (assuming D is not sufficiently high priority).

3.6 Related Work and Discussion

Web services are the most recent technological advancement in distributed information systems [Woh03, Pap05]. Therefore, web services related challenges could be understood

by considering how distributed systems evolved in the past. Web services were emerged to solve the network and system heterogeneity problems that the enterprise application integration (EAI) community faced for decades [Woh03, Aal02, Aal03a]. They hide the network and platform heterogeneities providing a uniform interface (WSDL) to describe services, a common communication protocol (SOAP) to send messages among services, and a directory (UDDI) service to publish and find services. In [Dou03], authors have argued that web services will play a major role in electronic data exchange and transaction processing systems. In [Ley02], authors illustrate how existing WSs are tailored to develop business processes over the Internet. Such applications need several web services to be integrated together, which implies proper coordination (in particular, control flow and dataflow) as well as message handling (sequencing and correlation) among participating web services to accomplish the business logic efficiently. Moreover, web service integration is intended to enable inter-organizational collaboration. Those coordinated activities are long-running (workflows, transactions) and require much more beyond invoke-response protocols and conventional transaction protocols such as two phase commit (2PC) are not suitable [Ley02, Yun98, Lit03].

Table 3.1 Comparison of Web Service Coordination Languages/Standards

	Mode of Operation	Data flow	Failure recovery	Quality of Service (QoS)	Web Services Based	Distributed Coordination	Easy Configurability /Workflow Development	Mobile-Domain Support
XPDL ¹	Activities invoke applications	No data conversion	Not-Specified	Activity attributes such as, duration, cost...etc	Not Specifically for web services	No	Partial	No
BPML ²	Activities implement actions	Data mapping from one activity to another	Compensation, Time-outs and event handlers	Not-Specified	Yes	No	Partial	No
BPEL4WS ³	Activities invoke Web services	Data containers	Compensation	Deadline and durations	Yes	No	Partial	No
Web Bonds⁴	Web services invoke other web services	Data adaptors	Partial	Availability, priority, expiry time	Yes	Yes	Yes	Yes (Using SyD middleware)

1 [Sha02], 2 [20], 3 [Woh03], 4 [Bal05a, Pra04c, Pra05]

Furthermore, currently individual WSs are stateless and do not have any provision to store state information for long-lived transactions/workflows [Aal03a, Pel03]T. Many languages, including WSFL [WSFL], WSCI [WSCI], WS-Coordination [WSC02], WS-Conversation, BPML, XLANG [Tha01], BPSS, and BPEL4WS [Woh03] have emerged as WS composition and coordination languages [Aal03c, Wee05]. Table 1 compares and contrasts characteristics of a cross section of these main languages with web bonds.

Coordination Primitives: First, we focus on the coordination primitives of these languages. These languages/standards propose various techniques for inter-linking different WS's together to form a composed web service application (web process). Web process consists of activities that are linked with participant web services. Links act as the communication channel, and all the communication handling need to be programmed by the developer (Table 1, Column 1). Among these techniques, WSFL proposed three

types of links: control links, data links, and plug links [Ley02]. “Control links connect the completion of one activity to the execution of another. Data link connection represents a data exchange between the two web services, and plug link represents the inherent client/server structure of a web service” [Ley02]. In BPEL4WS, a partner represents both the consumer (sequester) and the producer (supplier) web service [Wee05]. Partner link is associated with two WSDL port types of interacting web services. Partner link is bi-directional and it defines the shape of a relationship with a partner. Bi-directionality of partner links enables two services to exchange messages during the lifetime of the process instance. Since BPEL is based on IBM’s WSFL and Microsoft’s XLANG, BPEL partner links underpin WSFL’s control links, plug links and data link concepts. The key difference between BPEL4WS partner links and web coordination bonds is that web bonds allow dependency modeling through negotiation bonds while partner links act as a channel between two port types between two interacting services for data exchanges and invocations. Group dependencies and constraints need to be modeled using other language constructs. Therefore, partner link is used to directly model peer-to-peer conversational partner relationships.

The XML Linking Working Group proposed the XLink language, which is capable of establishing relationships between resources or portions of resources on the web. Currently this workgroup is not active. However, XML’s RDF [RDF04] proposes a similar idea. Both XLink and RDF “provide a way of asserting relations between resources” [RDF04]. In particular, RDF is a XML based meta language for representing information about resources in the World Wide Web. RDF is based on the idea of identifying things using Web identifiers (called *Uniform Resource Identifiers*, or *URIs*), and describing resources in terms of simple properties and property values. RDF allows establish relationship among entities of the resource based on its class definition schema and intended for programs to read and understand them [Eli05]. Such technologies have the potential to evolve as useful tools for WS-composition.

Semantic web community also proposes an ontology-based framework OWL-S (DAML-S) to enhance the web service infrastructure [Ver05, Bra03]. OWL-L proposes a new layer of metadata on top of WSDL so that services can be described and discovered semantically. Such enhancements should strengthen the integration and composition and provide automatic verification mechanism [Hul04]. Detailed discussion on semantic web service based composition is in [Ver05, Bru05, Bra03]

Formalizing Web Service Coordination Techniques: In [Ben02], authors have pointed out that lack of fundamental primitives for web service integration has resulted in plethora of products and standards. These standards are overlapping competing, and far from being complete. They require refinement, consolidation, standardization, and theoretical treatment to find a small yet powerful core set of threading primitives [Sta03]. In [Bru05], authors present a hierarchy of transactional calculi with increasing expressiveness. They start from a very small language in which activities can only be composed sequentially. Then, progressively introduce parallel composition, nesting, programmable compensations and exception handling. In [1], author discusses pros and cons of Petri nets and Pi calculus for web service conversion languages (WSCL) and illustrates fundamental differences between Petri nets and Pi calculus. A choreography language named CL [Bus05] is another noticeable effort towards formalizing web coordination. Following the approach of WS-CDL, in CL choreography contains a “global” definition of the common ordering conditions and constraints under which messages are exchanged within a conversation among collaborating services. In [Luc05], authors argue that three different mechanisms for error handling available in BPEL are not necessary in web service composition. They have formalized a novel orchestration language based on the idea of event notification as the unique error handling mechanism, and present a formal definition of three BPEL mechanisms in terms of their calculus. In [Coo05], authors propose a programming language which directly supports Web service

development that leverages XQuery for native XML processing, supports implicit message correlation and has high level calculus-style concurrency control. However, such developments are in very early stage and much remains to be done to find a web service “coordination theory.”

3.7 Summary

The next generation Internet applications will be various kinds of collaborative applications among heterogeneous, autonomous entities deployed over the web. Even if there are a variety of products and standards for web services composition, there is no fundamental framework to develop and deploy collaborative applications over web services. In this chapter, we have introduced the concept of web coordination bonds as an effort towards a fundamental set of primitives for web service coordination. Web coordination bonds enable web services to create and enforce interdependencies and constraints, and carry out atomic transactions spanning over a group of web entities/processes. We have demonstrated the concept of web coordination bonds as a capable framework to develop and deploy such collaborative applications with the required theoretical underpinning. We theoretically showed its modeling power is equivalent to the modeling power of extended Petri net. We also highlighted the expressive power of web coordination bonds by modeling various dependency scenarios. In the next chapter we further illustrate the on expressiveness of web coordination bonds by modeling a comprehensive set of benchmark workflow control flow patterns and distributed communication patterns.

CHAPTER 4

EXPRESSIVENESS OF WEB COORDINATION BONDS

In Chapter 3, we have presented the idea of web coordination bonds and proved that web bonds can model extended Petri-nets, and thus, are fundamentally capable primitives (Modeling power [Wee05, Pra05]. However, in practical terms, what matters most is their expressiveness or the suitability [Kie02]. As we have mentioned earlier, modeling power and the expressiveness are closely related terms. However, the subtle difference between two terms is that the former indicates the ability to design or model coordination (interaction) patterns whereas the latter denotes how efficiently and easily such patterns can be modeled. Therefore, modeling interaction patterns is a suitable benchmark for evaluating expressiveness [Aal03a, Aal03b].

Significant research work has been carried out by both academia and industry to identify interaction/ dependency patterns and adequacy of workflow languages to enforce such interaction patterns in web service coordination/choreography [Aal03a, Aal03b, Aal04, Ben02, Dus04, Bru05]. One of the perceptible outcomes of such analysis has been the identification of different categories of workflow control flow patterns, communication patterns, and resource sharing patterns. Among interaction patterns, workflow control flow patterns and distributed communication patterns capture essential requirements to model workflow dependencies [Gua98, Hua98, Pre02]. Therefore, here, we demonstrate the expressiveness of web coordination bonds by modeling a comprehensive set of benchmark workflow scenarios and distributed communication patterns. In addition, a comparative analysis is presented against corresponding BPEL and Petri-Net based constructs for aforementioned patterns.

4.1 Modeling Workflow Control Flow Pattern: Background

In this section we briefly discuss workflow control flow patterns, WS-BPEL and Petri net terminology that will be used in this Chapter. In [Aal03c], authors have gathered following six categories of control flow patterns that occur in workflows. Table 4.1, briefly describes a benchmark set of workflow control flow patterns.

Table 4.1: Workflow Control Flow Patterns

Category	Benchmark patterns	Description
Basic control flow	Sequence	An activity of a workflow is enabled after completion of another activity the same workflow.
	Parallel Split	AND split is a point in a workflow where control is passed to multiple paths and all paths are executed in parallel
	Synchronization	Synchronization is a point in a workflow where multiple control paths converge into a single control
	Exclusive Choice	XOR-Split is a point in a workflow where one of possible paths is selected.
	Simple Merge	XOR-merge is a point in a workflow where alternative branches get together without synchronization.
Advanced branching and Synchronization	Multi-Choice	A point in a workflow where one or several paths will be chosen based on some selection criteria
	Synchronizing-Merge	OR-merge is a point in a workflow where several control paths converge into a single control. If more than one path is active synchronization is required

	Multi-Merge	Multi-merge is a point where several branches merge without synchronization. Also, for each active path activity followed by merge will be executed in execution order.
	Discriminator	A point in a workflow where it starts the subsequent activity as soon as one of the incoming paths is completed and waits for other paths to complete and ignore.
Patterns involving multiple instances (MI)	MI without synchronization	For any workflow activity, multiple instances of that activity can be created. These activities are independent and do not need to synchronize.
	MI with prior design time knowledge	For any workflow activity, multiple instances of that activity can be created. These activities need to synchronize before starting subsequent activities of the workflow.
	MI with prior run time knowledge	For any workflow activity, multiple instances of that activity can be created. These activities need to synchronize before starting subsequent activities of the workflow. Difficulty here is that numbers of instances is not known at the design time.
	MI without prior run time knowledge	For any workflow activity, multiple instances of that activity can be created. These activities need to synchronize before starting subsequent activities of the workflow. It becomes more difficult due to the fact that numbers of instances is not known at the design time.
State-based patterns	Deferred choice	A point in a workflow where one of the several possible paths is chosen. However, deferred choice is different from XOR logic in that choice is made by the environment (user) not explicitly based on data. Once a particular path is chosen other branches are withdrawn.
	Interleaved parallel routing	A point in a workflow where set of activities executed in any order. Importantly, all the activities will be executed. Order is not known before runtime.

	Milestone	Milestone is a state based control flow pattern where an activity is enabled only if a certain state has been reached and still not expired. Therefore, to start an activity that has milestone control dependency it needs to wait for that specified state.
Structured patterns	Arbitrary cycle	A point in a workflow where some set of activities (paths) can be repeated several times.
	Implicit termination	A workflow needs to terminate when there is no other activity to perform (on other active activity and no other activity can be made active)
Cancellation patterns	Cancel activity	Enabled activity is removed from the workflow.
	Cancel case	This is an extended version of cancel activity where the whole workflow instance is removed

- i. *Basic control flow* patterns capture simple control flow such as sequence, AND split, and AND joint.
- ii. *Advanced branching and synchronization* capture patterns such as synchronous merge and multi merge require complex decision-making.
- iii, iv. *Structured and state based* patterns require analyzing current execution state of the workflow. and decisions are made accordingly. Such decisions are made at runtime.
- v. Patterns involving *multiple instances* need to manage (create and synchronize) multiple instances of workflow activities during the execution of the workflow.
- vi. Finally, *cancellation patterns* need workflow to remove one or more activities or dismantle the whole workflow during the execution.

4.1.1 Business Process Execution Language for Web Service (WS-BPEL)

BPEL4WS [Alo04, WSCI] (Business Process Execution Language for Web Services) is a process modeling language developed by IBM, Microsoft, and BEA. It supersedes XLANG (Microsoft) and WSFL (IBM) and built on top of WSDL. BPEL defines activities as the basic components of a process definition. Structured activities prescribe the order in which a collection of activities take place (Table 4.2). Ordinary sequential control between activities is provided by *sequence*, *switch*, and *while*. Concurrency and synchronization between activities is provided by *flow* structure. Nondeterministic choice based on external events is provided by *pick*. In BPEL, process instance-relevant data (containers) can be referred to in routing logic and expressions (receive, send). It also defines a mechanism for catching and handling faults similar to common programming languages such as Java. One may also define a compensation handler to enable compensatory activities in the event of actions that cannot be explicitly undone.

Table 4.2: BPEL Primitives [WSCI]

BPEL-Primitives	Functionality	BPEL-Primitives	Functionality
<sequence>	One after the other	<reply>	Send msg to partner as response to <receive> <i>other</i>
<flow>	Parallel	<assign>	Manipulate variables
<pick>	Choose by inbound message	<wait>	For duration / until time
<while>	Iteration	<terminate>	End the process
<scope>	Nest, with declarations and handlers, synchronize <i>communication</i>	<compensate>	Run compensation handler
<invoke>	Send msg to partner; possibly receive response	<empty>	Do nothing
<receive>	Accept msg from partner	<throw>	Exit with fault to outer scope

BPEL Partner Links: The concept of partners is used to define two web services that are to be invoked as part of the process. It is based on two elements: a) partner link type: it contains two port types, one for each of the roles in the partner entry (i.e., one port type is the port type of the process itself, the other one is the port type of the service being invoked), b) partner Link: the actual link to the service. This is where the actual assignment to a binding is made (outside the scope of BPEL). Bi-directionality of partner links enables two services to exchange messages during the lifetime of the process instance.

5.1.2 Petri-net

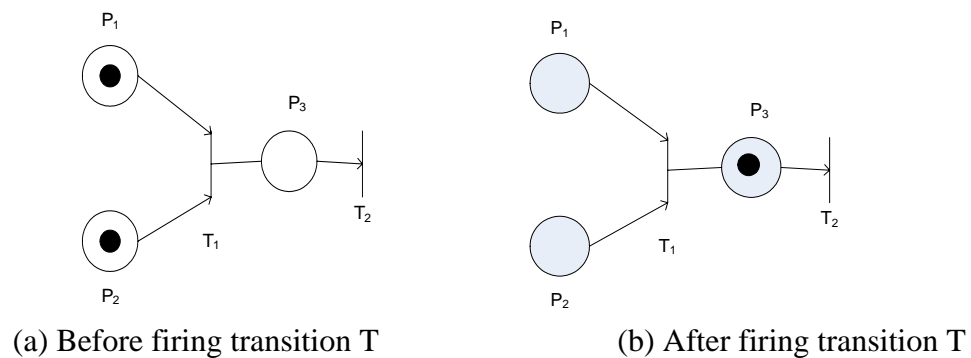


Figure 4.1: Petri-net model

Petri-net is one of the widely adopted tools for concurrent process modeling. Petri-net modeling has been developed on three fundamental primitive concepts: tokens, places, and transitions.

Tokens: dots that move between places.

Places: represents “states” of system based on the distribution of tokens.

Transitions: A transition has Zero or more input arcs coming from input places and zero or more output arcs going to output places. Transition is enabled if and only if there are one or more tokens in all input places. Enabled transition fires by removing one token from each input place and depositing one token in each output place.

Modeling power of Petri-net is equivalent to the modeling power of Turing machine [Age74]. Thus it has sufficient modeling power to model any computable function. Extended versions such as color Petri-net and timed Petri-net have been proposed for easy usage of the concept. However, fundamental capabilities remain the same. Extensive discussion on Petri-nets is in [Mur89]. Currently, significant amount of interest has been shown in modeling workflows and distributed computing scenarios over web services based on Petri-net modeling [Aal04, Aal02, Ben03].

Reminder of the Chapter discusses how to model these patterns using web coordination bond highlighting corresponding BPEL and Petri net-based implementations. We compare and contrast Petri-net, BPEL and web coordination bonds implementation alternatives.

4.2 Modeling Workflow Control Flow Patterns Using Web Coordination Bonds

Different workflow models have different expressive capabilities to enforce these control flow patterns [Aal03a]. However, analysis shows that none of them is comprehensive enough [Aal03a]. Table 4.3 shows a pattern-based analysis of BEPL, Petri-Net, WSCI, and Web Coordination bonds (Here, “+” implies direct support, “-“ implies no direct support, and “+/-“ implies direct support with some restrictions). As shown in Table 2, web bond artifacts have enough expressive power to enforce these control flow patterns

directly. Remaining sections of this Chapter discuss issues related to modeling these interaction patterns and reason out why web coordination bond is a better candidate.

Table 4.3: Support for workflow control patterns in different web service composition languages and standards [Wfp03, Aal02, Aal03a]*

Pattern	Web Bond	Standard/Product		
		Petri Net(Basic+ High level)	BPEL4WS	WSCI
1. Basic Control: Sequence	+	+	+	+
Parallel Split	+	+	+	+
Synchronization	+	+	+	+
Exclusive Choice	+	+	+	+
Simple Merge	+	+	+	+
2. Advanced Branching & Synchronization: Multi Choice	+	+	+	-
Synchronizing Merge	+	-	+	-
Multi Merge	+	+	-	+/-
Discriminator	+	-	-	-
3. Structural: Arbitrary Cycles	+	+	-	-
Implicit Termination	+	-	+	+
4. Multiple Instances: MI without Synchronization	+	+	+	+
MI with a Priori Design Time Knowledge	+	+	+	+
MI with a Priori Runtime Knowledge	+	-	-	-
MI without a Priori Runtime Knowledge	+	-	-	-
5. State based: Deferred Choice	+	+	+	+
Interleaved Parallel Routing	+	+	+/-	-
Milestone	+	+	-	-
6. Cancellation: Cancel Activity	+	+/-	+	+
Cancel Case	+	-	+	+

* We have taken column 2, 3 and 4 from ref [Aal03a]

4.2.1 Basic Control Flow Patterns

Basics control flow patterns capture simple split and join constructs. *Sequence*, the simplest of basic control flow patterns, requires an activity of a workflow to be enabled directly after the completion of another activity of the same workflow. *Parallel split* and *exclusive choice* dictates the workflow activity to split the control to multiple paths or pass the control to exactly one of possible paths respectively. *Synchronization* pattern requires that multiple control paths converge into a single control whereas *simple merge* requires alternative branches get together without synchronization. These constructs are relatively easy to implement and almost all the workflow models have mechanisms to support them (Table 4.2). Parallel split and simple merge constructs have being presented in this section. Implementations of other basic control flow patterns are in [Pra04c].

Parallel split (AND-Split): AND split is a point in a workflow where control is passed to multiple paths and all paths are executed in parallel (Figure 4.2a).

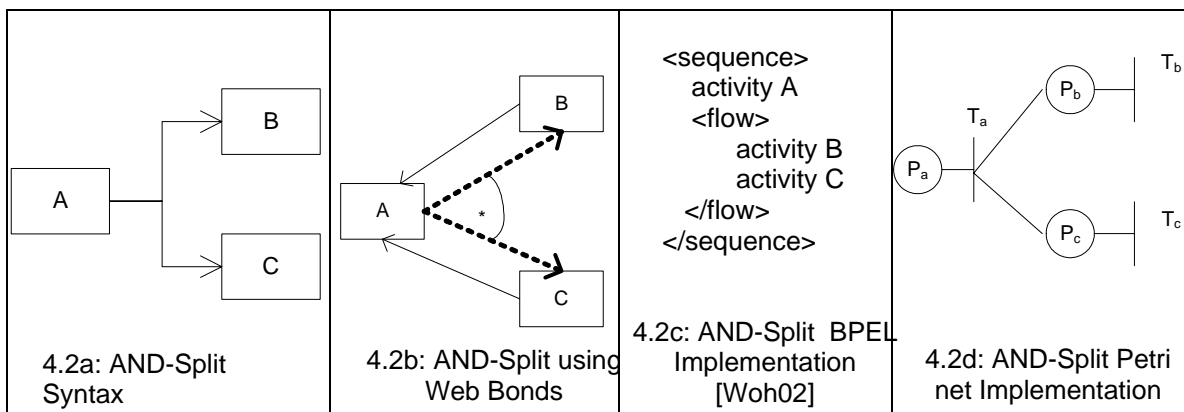


Figure 4.2: Parallel Split

Order of execution: $A \rightarrow [B, C]$ or $A \rightarrow [C, B]$

Implementation: Both, B and C , are to be executed in parallel once A is completed. It can be captured by creating subscription bonds from A to B and C (Figure 4.2b). These subscription bonds make sure that control is passed to both B and C simultaneously after the completion of A . Negotiation bonds from B , C to A are required to ensure that B and C can be executed only after A is completed.

BPEL enforces parallel split using flow activity control after the completion of A (Figure 4.2c). In Petri-net implementation, transition T_a represents the activity A of the workflow. When T_a fires, it puts a token each in places P_b and P_c enabling transitions T_b and T_c (Figure 4.2d) simultaneously.

Simple Merge (XOR merge): XOR-merge is a point in a workflow where alternative branches get together without synchronization.

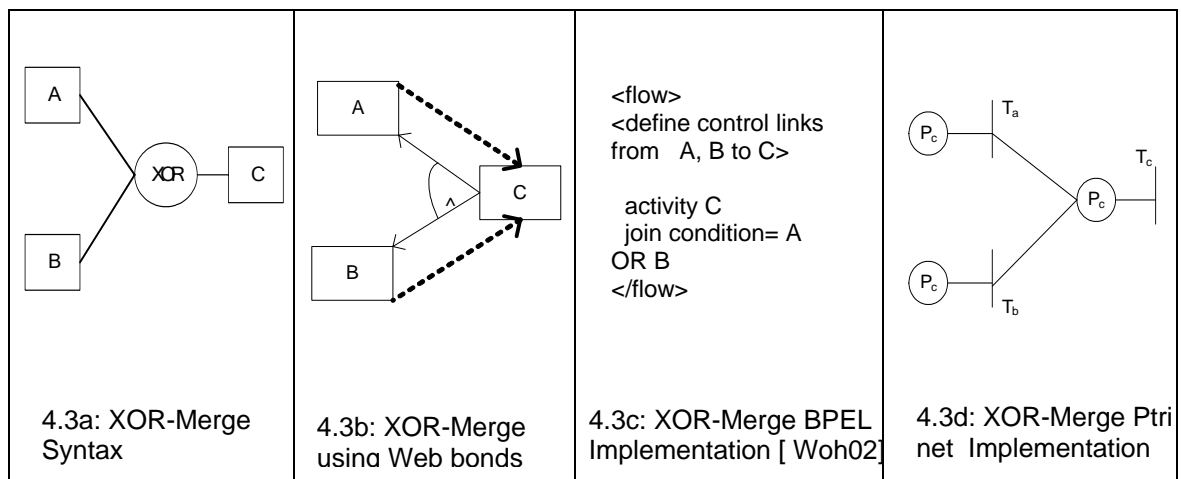


Figure 4.3: Simple Merge

Order of execution: AC, BC

Implementation: Complexity of XOR-merge is reduced due to the assumption that alternative threads A and B do not execute in parallel. Construction shown in Figure 4.3a implements the simple merge using web bonds. Negotiation bonds from C to A and B with XOR logic make sure that C will be executed only if one of A and B are active.

BPEL models simple merge by having control links from A , B to C and evaluating ‘OR’ join condition between bonds (Figure 4.3c). Corresponding Petri net construct is shown in Figure 4.3d. This is valid construct due to the assumption that either T_a or T_b gets fired placing only one token in place P_c . However, such assumptions may not be realistic especially in distributed settings. Relaxation of this assumption leads to advanced synchronization patterns such as Multi merge and Sync merge that will be discussed in following section.

4.2.2 Advanced Synchronization Patterns

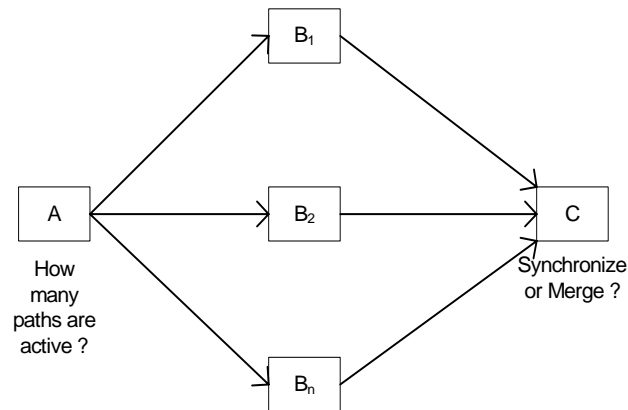


Figure 4.4: Advanced Synchronization

As shown in Figure 4.4, in advanced synchronization models, problem arises as the split node can activate m out of n paths ($0 \leq m \leq n$). When it comes to the synchronization,

synchronization node needs to know which paths to be synchronized. In some cases, synchronization needs to be done based on different merging criteria [Kie02]. Thus, synchronization is a significant issue in workflow modeling and has gained considerable attention [Bar05,Gor05, WSCI, Jan03, Wee05]. There are four advanced synchronization patterns: *Multi choice*, *Synchronous merge*, *Discriminator*, and *Multi merge*. Multi choice is the split of control to one or several paths based on some selection criteria. Three synchronization patterns; Synchronous merge, Discriminator, and Multi merge layout different rules of merging control flow.

Multi choice is a simple construct to implement and many workflow technologies have direct support. BEPL implements the multi choice by using switch-case construct or using partner links with OR logic embedded [WSCI]. Web bonds enforce multi choice by having subscriptions bonds from the split node to destination nodes with OR logic embedded. Evaluation conditions need to be specified during the bond creation time. Petri net enforces this logic by having AND-split followed by XOR-split. Synchronization patterns are hard to model. Here, we discuss synchronization patterns in detail.

Synchronous merge (OR - Merge) (Figure 4.5a): OR-merge is a point in a workflow where several control paths converge into a single control.

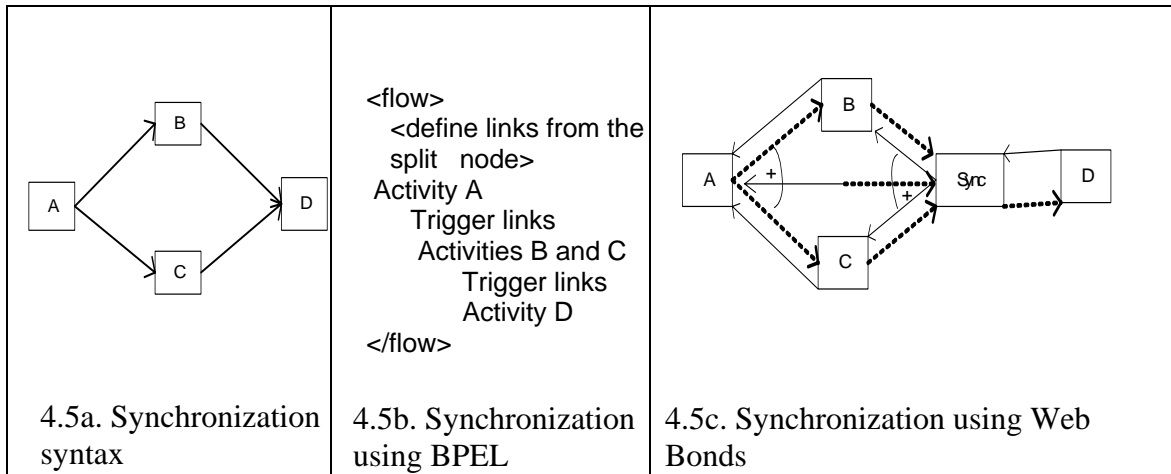


Figure 4.5:Synchronization Pattern

Order of execution: ABD, ACD, ABCD, ACBD.

Implementation: In Synchronous merge, if more than one path is active then these paths need to be synchronized. Otherwise only merge takes place. Main difficulty with synchronization is to decide when to synchronize and when to merge. As shown in Fig 9c, this difficulty can be eliminated by creating a subscription bond from activity A to “Sync” activity. This subscription bond transfers data pertaining to the split of control at A. Then, based on that data “Sync” waits for all the active paths before activating the subscription bond from “Sync” to D. A Negotiation bond from “Sync” to A is required because “Sync” must start its activities after the completion of A.

It is not easy to model such patterns using Petri-net based models as Petri-net supports only XOR-join or AND-join directly [Aal03a]. There are several alternatives solutions to this problem.

1. Split node informs the synchronization node which paths to synchronize (as we have used in web bond based implementation).

2. Activate all the paths from split node with either true or false tokens.

Synchronization node can synchronize the true paths and ignore the false paths.

For the first solution, the designer has to put some extra logic to send information from split-node to join node, and also the join node to process it. Such logic is not available in Petri-net. In the second solution, the designer has to extend the Petri net model to accommodate true/false tokens. BEPL support this construct as it allows control links to pass true/false tokens via control links. This method is known as the dead path elimination [WSC1] (Figure 4.5b).

Multi Merge (Figure 4.6a): Multi-merge is a point where several branches merge without synchronization. Merge activity will be instantiated several times.

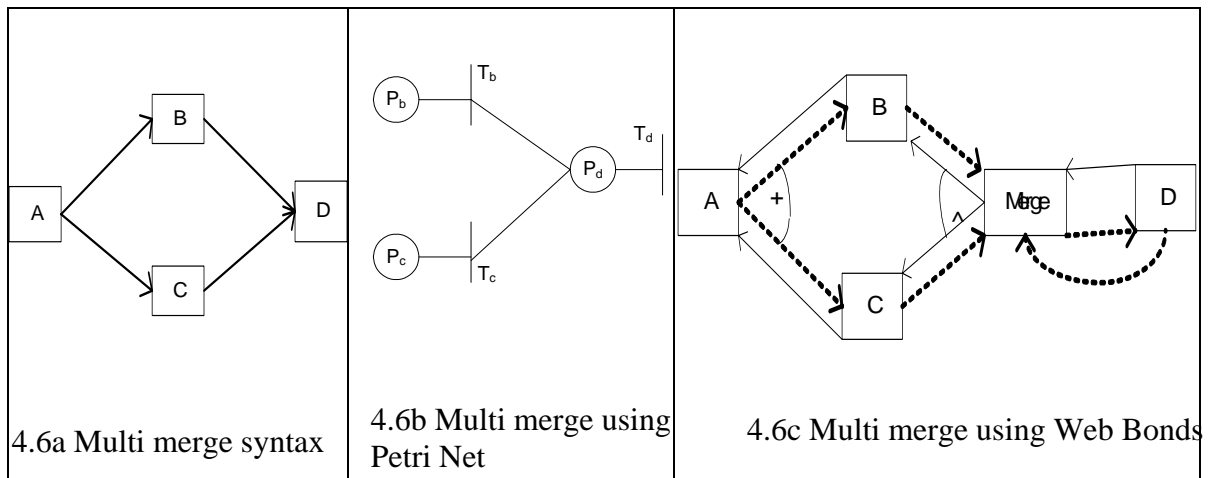


Figure 4.6: Multi Merge

Order of execution: ABD, ACD, ABDCD, ABCDD, ACDBD ACBDD

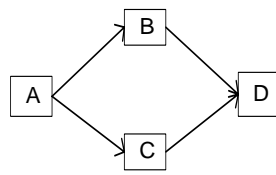
Implementation: In this construct, activity *D* will be activated several times based on number of active paths. This can be enforced using the bond structure shown in Figure 4.6c. *A* is the split point with OR split. “Merge” has to execute *D* as many as number of active control paths. This can be implemented as follows.

Merge has negotiation bonds with OR logic with all incoming paths. Once it receives control from one of its incoming paths, “*Merge*” makes a copy of its out going bonds. Then removes all the bonds related to the currently active path. Then it triggers the subscription bond from merge to D . Once D finishes its execution D triggers the subscription bond back to “*Merge*”. At this time, “*Merge*” reinstates copied bonds back and repeats the same procedure for all other incoming controls.

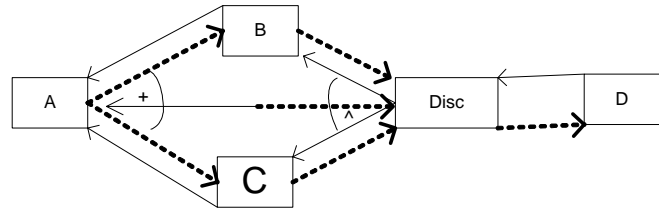
BPEL does not have direct construct because the designer has to keep track of if an instance of D is running and wait for it to finish before stating another. Otherwise it has to create a new instance of D , which is not intended here. Petri-net nicely capture as T_d can be fired only when there is a token in place P_d and it is ready to fire. T_d becomes ready once it completes the current execution of T_d .

Unlike synchronization, “*Merge*” create instances of D each time it receives control from an active path. Therefore, “*Merge*” does not need to know information about active paths in advance. In synchronizing D is executed once. Here, in Multi-merge activity after multi-merge is executed several times based on number of active paths. We can have a control pattern between those two extremes where activity D is executed once but it can be started as soon as one of B or C is completed. This is called the *discriminator*, the next pattern.

Discriminator (Figure 4.7a): A point in a workflow where the activity is started as soon as one of the incoming paths is completed. Then it waits for other paths to complete and ignores the control.



4.7a Discriminator syntax



4.7b Discriminator using web bonds

Figure 4.7: Discriminator pattern

Order of execution: ABCD, ACBD, ABDC, ACDB

Implementation: In discriminator construct, activity “Disc” waits for the control from one of the incoming paths and activates *D*. After that it waits for remaining paths for the control and ignores them. This can easily be enforced by creating a separate activity “Disc” with the bond structure as shown in Figure 4.7b. Negotiation bonds from “Disc” to *B* and *C* with OR logic ensure that “Disc” can get control from several paths. However, in this case, it has to wait for only one specific control path. This information needs to be sent by *A* or “Disc” has to decide it based on runtime data. Former can be enforced by having a subscription from *A* to “Disc”. However, latter is workflow designer’s responsibility. Once “Disc” receives control from the desired path, it activates the subscription bond from “Disc” to *D*. Subsequent invocations to “Disc” through subscription bonds from incoming paths will be ignored because the subscription bond from “Disc” to *D* has already been fired.

Both BPEL and Petri net do not support this construct. As pointed out in [Wee05], BPEL join constructs are evaluated once all links have their logical value. However, this case requires first positive link to be identified and precede the execution of the workflow. Other links need to be ignored. In case of BPEL it is workflow designer’s responsibility to incorporate such logic. Colored or timed Petri-net can be used to model this pattern.

However, the workflow designer has to incorporate extra logic to identify the proper tokens to enforce the discriminator and discard other tokens.

M out of N: This can be deduced from construct for synchronous merge with m paths out of N . In this case, “*Disc*” waits for M incoming branches to be completed before starting the next activity and waits for other incoming branches and ignores them.

4.2.3 Patterns Involving Multiple Instances (MI)

Multiple instance patterns require workflow activity to instantiate several instances of the activity. In some situations, these instances need to be synchronized under various conditions before proceeding to the next activity of the workflow. Four patterns involving multiple instances have been identified [Aal03c]: a) Multiple instances without synchronization, b) Multiple instances with prior design time knowledge, c) Multiple instances with prior runtime knowledge, and d) Multiple instances without prior runtime knowledge. To facilitate multiple instance patterns, workflow activity should support multiple instantiation. Table 2 Illustrates how to model these multiple instance creation patterns using web bonds highlighting corresponding BPEL and Petri net alternatives.

Multiple instances without synchronization: Among those four patterns, this is the simplest as it does not need to synchronize with instances. Therefore, any activity can instantiate as many instances as required and transfer the control to the next activity. The next activity does not need to wait on all the instances to be finished before starting its execution. In fact, this is similar to sequence in terms control flow structure.

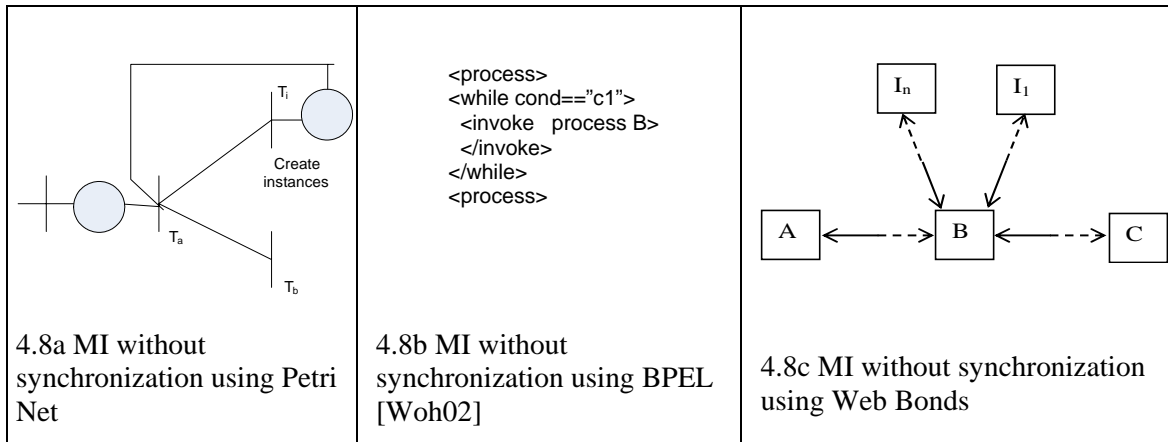


Figure 4.8: MI without Synchronization

Figure 4.8c shows the bond structure to enforce this pattern. Activity *B* will create multiple instances of it and then passes the control to *C*. This can be achieved by set of subscription bonds from *B* to each of its instances. This enables instances to be created with suitable initial data set. As soon as instances are created, *B* triggers the subscription bond from *B* to *C* and passes the control to *C*. At this time, instances may active and running. Most of the workflow models support this construct. Both BPEL and Petri-net support this construct directly. BPEL spawns as many instance as required using a while loop (Figure 4.8b).

Multiple instances with prior design time knowledge: In this case, synchronization is required but number of instances is known at the design time. All three modeling techniques support this construct (Figure 4.9). Here, the control flow logic is similar to AND-Split followed by AND-Join.

Implementation: As number of instances is known at the design time, placeholders for them are created at the design time. This can be enforced through parallel split followed by synchronize merge. Fig. 13c shows the bond structure to enforce this control flow.

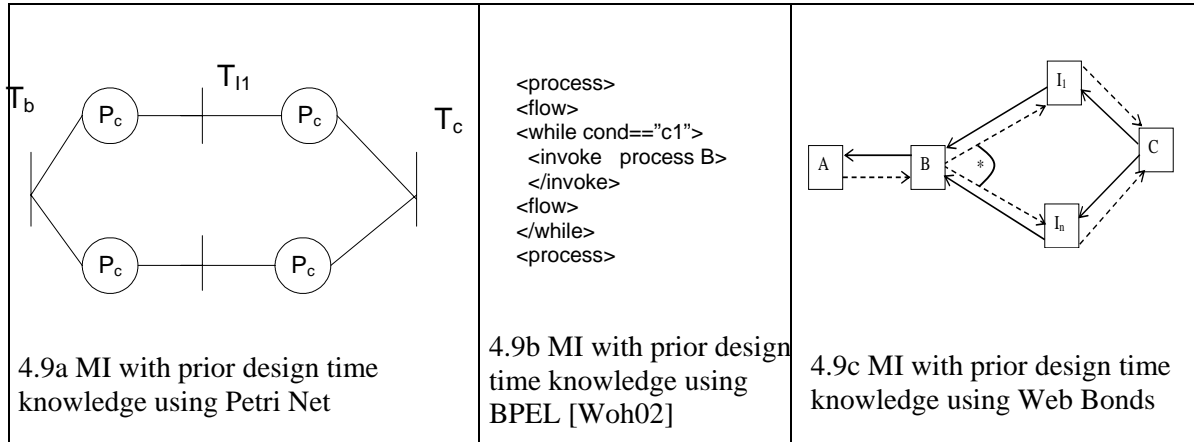


Figure 4.9: MI with prior design time knowledge

Multiple instances with or without prior runtime knowledge: These patterns are hard to model. Designer of the workflow is not aware of number of instances at the design time. As it is a runtime parameter designer cannot model them using place and transitions in Petri-net. Therefore, the designer has to come up with the logic to control and keep track of number of instances and synchronize them. Such modeling is difficult and need considerable effort. Both BPEL and Petri net do not directly support this construct [Men04, Aal02]. Programming language techniques outside of Petri-net or BPEL core primitives are required (Table3, Columns 1, 2). Keeping a counter and updating it when instance are spawned and terminated would be a one simple solution (Table 3, Column 2). However, web coordination bonds enable such dynamic modeling due to it ability handle message based as well as state based synchronization and the dynamic nature.

Subscription bonds and negotiation bonds keep track of instances and synchronize them accordingly.

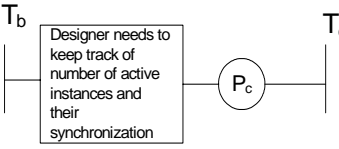
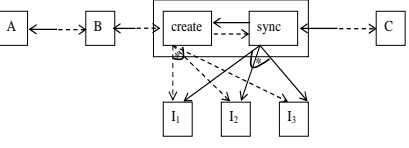
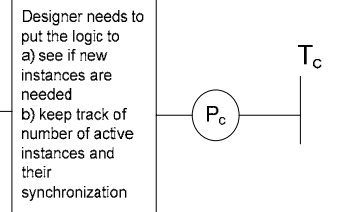
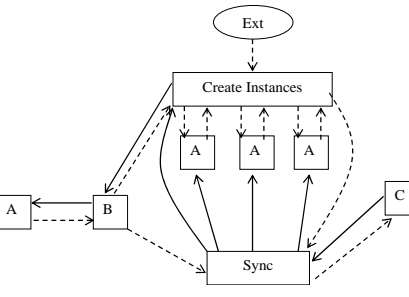
MI with prior runtime knowledge: For any workflow activity, multiple instances of that activity can be created. These activities need to synchronize before starting subsequent activities of the workflow.

Implementation: As number of instances is not known at the design time, most of the workflow models cannot enforce this construct. Due to the dynamic creation and deletion facility of web bonds, this can easily be enforced using web bonds. To enforce this control we introduce a new node which is capable of creating and synchronizing instances (Table 4.3, Row 1, Column 3). Here, activity *B* passes the control to “*create*” sub-activity with instant creation parameters. Subscription bonds (with AND logic embedded) will be created with each instance at runtime. At the same time, it makes sure that the sub-activity “*sync*” creates negotiation bond with each instance. This is achieved through the subscription bond from “*create*” activity to “*sync*” activity. This subscription bond passes all the instance related information to “*sync*” and then “*sync*” creates negotiation bonds with each instance at runtime. Having negotiation bonds to each instance, “*sync*” activity ensures that it waits for all instances to be finished before passing the control to *C*.

MI without prior runtime knowledge: For any workflow activity, multiple instances of that activity can be created. These activities need to synchronize before starting subsequent activities of the workflow. Unlike previous case, here, number of instances is not known before runtime. ***Implementation:*** This is one of the most difficult controls to be enforced. web bonds can enforce this relatively less difficulty. In order to accomplish

this we can create the bond structure as shown in Table 4.3, Row 1, Column3, web bond based implementation. “*Create Instance*” activity is capable of spawning new instances. All the instances must be synchronized before activating activity *C*. In order to achieve this *C* has a negotiation bond with “*Sync*”. When “*Create Instance*” activity creates a new instance, “*Sync*” activity adds a new negotiation bond to that instance dynamically. This can be achieved by having two subscription bonds from “*Create Instance*” activity to new instance and “*Sync*” activity with AND logic. With this construct, “*Sync*” can only complete its activity once all the instances are done. “*Ext*” is an external activity that may trigger “*Create Instance*” activity to create new instances.

Table 4.4: Patterns Involving Multiple Instances

Petri-Net based	WS-BPEL [Wee05]	Web Coordination Bonds
<p>a) MI with prior runtime knowledge</p> 	<pre> moreInstances:=True i:=0 <while moreInstances OR i>0> <pick> <onMessage StartNewActivityA> invoke activityA i:=i+1 </onMessage> <onMessage ActivityAFinished> i:=i-1 </onMessage> <onMessage NoMoreInstances> moreInstances:=False </onMessage> </pick> </while> </pre>	
<p>b) MI without prior runtime knowledge</p> 	<p>No direct support</p>	

4.2.4 State Based Patterns

State based patterns require control path of the workflow to be decided based the current execution status of the workflow. Here, we illustrate how to enforce these constructs using web coordination bonds. Also, corresponding BPEL and Petri net constructs have been discussed.

Deferred Choice (Figure 4.10a): A point in a workflow where one of the several possible paths is chosen. However, deferred choice is different from XOR logic in that choice is made by the environment (user) not explicitly based on data. Once a particular path is chosen other branches are withdrawn.

Implementation: As shown in Figure 4.10a, *B* is the deferred choice point where several alternatives are offered and only one is chosen. Unlike XOR split, here, alternatives are offered to the environment and upon selection of the appropriate control path, other alternatives are withdrawn. This can be achieved with bond structure shown in Fig. 14b. “*Ext*” is the workflow activity that receives external inputs for the deferred choice. When “*Diff*” is active, “*Ext*” can select either *B* or *C* through the subscription bond from “*Ext*” to “*Diff*”. Negotiation bond from “*Diff*” to “*Ext*” make sure that “*Diff*” can be invoked only if “*Ext*” sends its selection. This invocation triggers subscription bond with XOR logic to *B* and *C*. Only one bond will be selected and other bond will be deleted at runtime. Deletion makes sure that other alternatives are withdrawn.

BPEL implements this construct using pick activity. Pick activity waits for the appropriate message before passing the control. As shown in Figure 4.10c, upon receipt of

the message, says C , it picks the activity C and execute. Corresponding Petri net implementation is shown in Figure 4.10d, once T_a fires; it puts a token in place P_a . Then, whenever, place T_{ext} has a token it can fire either T_c or T_d . T_{ext} gets a token when external even T_{ext} fires.

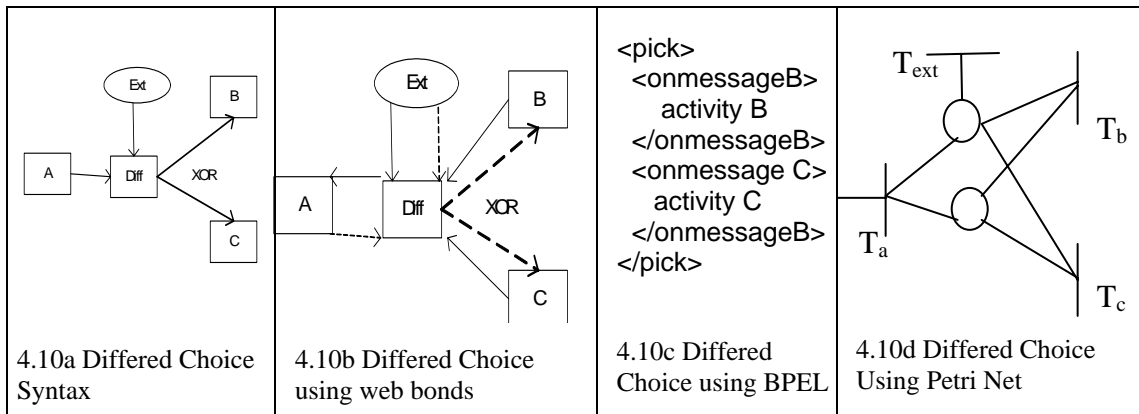


Figure 4.10: Differed Choice

Milestone: Milestone is a state based control flow pattern where an activity is enabled only if a certain state has been reached and still not expired [Aal03c]. Therefore, to start an activity that has milestone control dependency it needs to wait for that specified state. For example, as shown in Figure 4.11a, activity C is enabled if activity A has been completed, hence M has the control, and B has not been completed, hence the control is still in M . In other words, control has been released from A and has not been consumed by C yet. This situation can easily be modeled using middle activity M [Aal03c].

Implementation: This is difficult control to enforce because there is a race condition among activities and the execution of some activities may disable others. Most workflow systems do not have automatic way of disabling and enabled activity. However, milestone can easily be enforced using the power of negotiation bonds as shown in Figure 4.11b. C has a negotiation

bond to M . This means that C can only be done if M is completed. In this case, M is completed if M has the control. In addition, M has a subscription bond to inform the arrival of control to C . Negotiation bonds from M to A and B are also required to enforce dependencies of M to A and B to M .

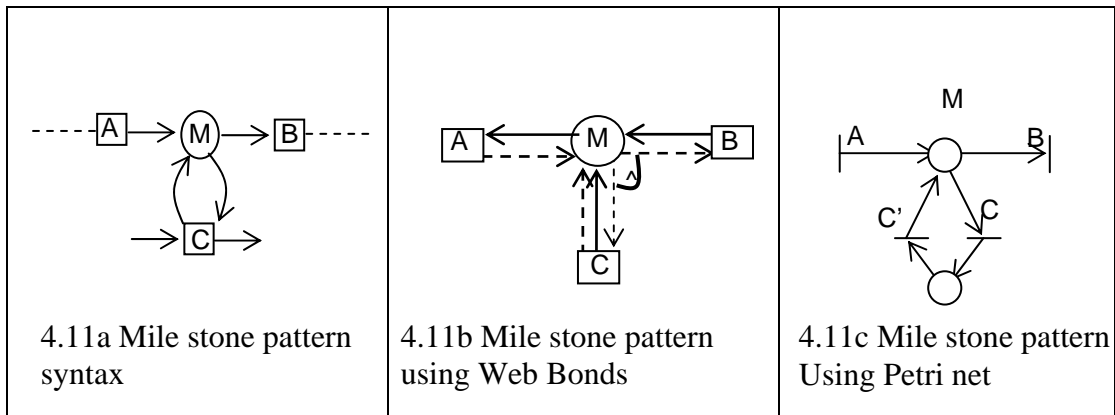


Figure 4.11: Mile stone pattern

Petri-net has direct support for milestone and all other state based construct because original Petri-net concepts are based on representing state of different activities. As shown in Figure 4.11c, once M has a token it enables both B and C . But if C gets the control it just fire it and then via dummy transition C' , C puts the control back in M . If M gets the control, then C is disabled and it is no longer available to fire. This is exactly the behavior expected from the milestone pattern. Once aging, BPEL does not have proper constructs available as the designer need to keep track of a) The availability of control at b) Invoke either C or B and, c) If C is invoked place the control back to M . In [Wee05], authors presents a work around BPEL solution to milestone using while and pick activities.

Interleaved parallel routing (Figure 4.12a): A point in a workflow where set of activities are executed in any order. Importantly, all the activities will be executed. Order is not known before runtime.

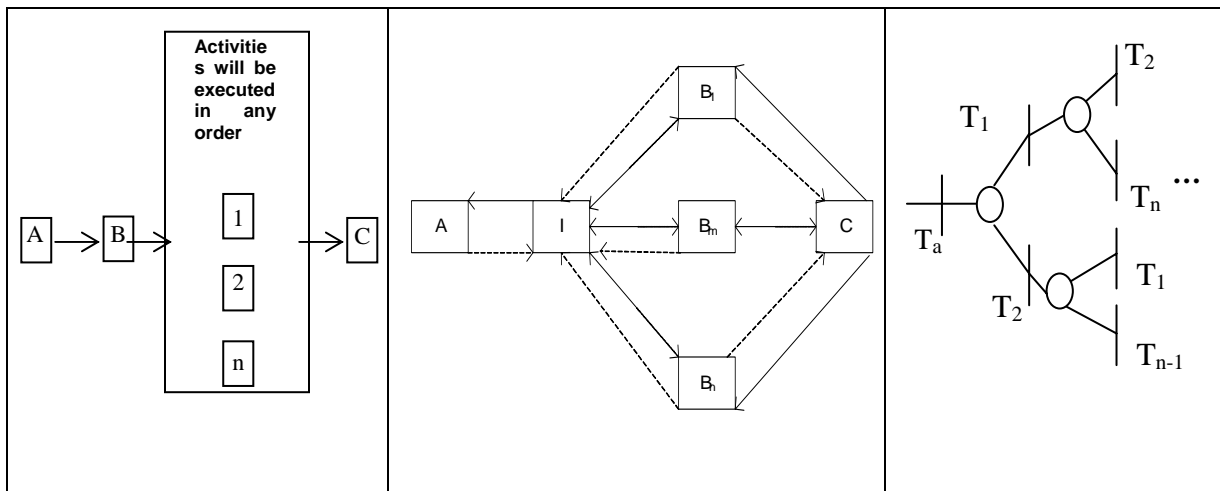


Figure 4.12: Interleaved Parallel Routing

Implementation: Interleaved parallel routing is one of the difficult control patterns to be modeled. Petri net provides a satisfactory solution with the cost of having extra node (place) that does not belong to the original workflow. Using web bonds an explicit “interleaver” construct can be modeled using the bond structure as shown in Figure 4.12b. Operation of the “interleaver”, I , is as follows.

I has three subscription bonds to each of $B_1 \dots B_n$ XOR logic. Once I receives the control from A , it selects one of the outgoing paths, say B_m . Upon selection of that bond, I makes a copy of the selected bond to a temporary location. Then the bond will be removed from the original group. In this case, two bonds with XOR logic will remain after the deletion of first bond. Finally, copy of the bond will be executed by enabling selected path (In this case B_m

will be enabled). Upon completion of the selected activity (B_m), it sends the control back to I and the activity C . This will enable I again. Then I will select one of existing paths and follow the same procedure. However, C will not be enabled until activities $B_1 \dots B_n$ are completed in any order. This is enforced by having negotiation bonds from C to each of B_1 through B_n .

Petri net based implementation of this pattern is shown in Figure 4.12c. Tree like structure ensures that the section of each activity is arbitrary. However, when there are many workflow nodes, tree becomes very large. BPEL does not have direct constructs to implement this pattern. In [Wee05], authors present a work around solution. In their solution, a container, which has exclusive access, has been implemented and each activity gets access rights to the container randomly. An activity currently holding the container will be executed. Upon release of the container, another activity acquires the access rights.

4.2.5 Structural Patterns

There are two types of workflow structure based patterns: arbitrary cycle and the implicit terminator.

Arbitrary Cycle (Figure 4.13): A point in a workflow where some set of activities (paths) can be repeated several times.

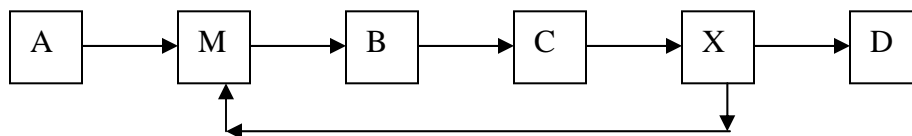


Figure 4.13: Arbitrary cycle

M= Merge, X= XOR

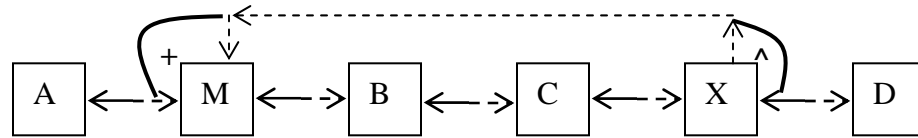


Figure 4.14: Arbitrary cycle using web bonds

Implementation: Arbitrary cycle is relatively easy construct to model (Figure 4.14). Activity *M* merges two paths from *A* to *M* and *X* to *M*. *X* is the activity which creates the arbitrary cycle. From *X*, subscription bond with XOR logic puts the control in cycle path or normal path. Merge activity has two negotiation bonds with XOR logic to *A* and *X*. They make sure that merge is active if either activity *A* or activity *X* is completed. *X* and *M* can be places in any arbitrary location of the workflow with above bond structure that supports the arbitrary cycle.

XOR split of Petri net can be used to direct the control to any location of the workflow that enables activities to be repeated. BPEL does not support this construct as it does not have jump instruction. While loop cannot be used as it enables repetition with definite entry and exit points [Wee05].

Implicit terminator: A workflow needs to terminate when there is no other activity to be performed.

Implementation: Web bonds, by their nature, make sure that workflow activities do not require such explicit final node because activity itself acts as an implicit terminator. If an object in a workflow does not have any live bonds (both in coming and outgoing) it acts as an implicit terminator. BPEL follows a similar logic using flow constructs and links. Activities can have sink activities which are not source for any link without requiring one unique terminating node [Wee05]. However, in Petri-net, it not easy to implement this as

the designer has to keep track of running threads before completing the workflow [Aal02].

4.2.6 Cancellation Patterns

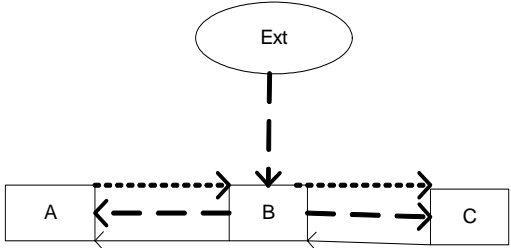
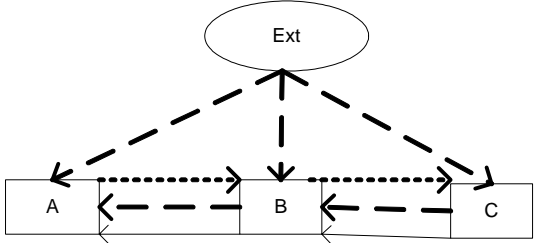
Cancellation patterns are difficult to realize and different application will have different requirements. First, cancel activity and cancel case will be discussed then we explain the logic behind cancellation using web bonds by implementing the cancellation of meeting scenario.

Cancel Activity (Table 4.5, row 1): Cancellation of an activity requires it to be removed from the workflow. There are several possible ways that this can be implemented using web bonds. Simplest method is to introduce an external activity “Ext” having a subscription bond to another activity that may be cancelled in the future. In this case, once “Ext” triggers the subscription bond, it will disable the activity *B*. When *B* is cancelled, it deletes (invalidates) all outgoing bonds attached to it. This will virtually remove the activity from the workflow. However, cancellation of an activity may trigger another set of cancellation/compensation activities of the workflow. As shown in web bond based implementation subscription bonds from *B* to *A* and *C* enforce such dependencies. For example, cancellation of an airline reservation will prompt hotel and car rental reservations to be cancelled. Such scenarios have to be identified during the design time. In fact, this is true for cancel case pattern also.

Cancel case (Table 4.5, row 2): This is an extended version of cancel activity where the whole workflow instance is removed. Cancel case is an extension to the cancel activity. Cancel case is relatively easy to implement using web bonds. In order to accomplish this we can have an external activity “Ext” which has subscription bonds to all activities in the

workflow with AND logic. Once “*Ext*” triggers subscription bonds, each activity deletes all active bonds attached to it. This will virtually dismantle the workflow.

Table 4.5: Cancellation Patterns

WS-BPEL [Wee05]	Web Coordination Bonds
<p>Terminator activity</p> <pre><scope> terminate A trigger appropriate compensation and fault handling </scope></pre>	
<p>terminate process <...> terminate the whole process (whole workflow or the sub process of the workflow)</p>	

4.3 Modeling Communication Patterns

Table 4.6: Communication patterns (Values for column 2 have been taken from reference [Wee05])

Pattern	WS-BPEL	Web Coordination Bonds
Synchronous 1. Request-Reply	+	+
2. One way	+	+
3. Polling	+	+
Asynchronous 1. Message passing	+	+
2. Publish/Subscribe	-	+
3. Broadcast	-	+

Message interaction among different entities of a distributed system is vital to its flexibility [Wee05]. Two basic distributed communication paradigms are synchronous and asynchronous communication. Synchronous communication needs the message sender to halt its process until it receives an acknowledgement or data from the receiver whereas asynchronous does not have such requirement. Any fundamental framework that facilitates composing applications over distributed components/objects must support both types of communication. As shown Table 4.5 web bonds have expressive capabilities to model these communication patterns directly. However, BPEL does not directly support asynchronous communication constructs. In this Section we illustrate how web bonds can be used to enforce different types of synchronous and asynchronous communication patterns.

4.3.1 Synchronous Communication

In synchronous messaging, message sender halts its execution until it receives the reply from the receiver. There are three different synchronous messaging patterns: request/reply, one way, and polling. In case of *Request/reply* scenario, sender expects the message receiver to send data/control to the sender while *One way* scenario expects the receiver to acknowledge the receipt of the message. Finally, *Polling*, allows the sender to continue its processing to while it is waiting for the reply. However, sender polls in regular intervals to the receiver to check the availability of results. Here, we illustrate the Request/Reply scenario. BPEL directly supports all the synchronous messaging [Wee05].

Request/Reply

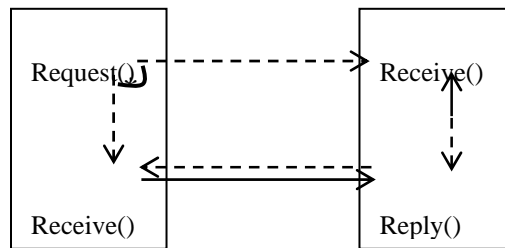


Figure 4.15: Reply/Request

Reply request scenario needs sender to halts its operation until it receives data from the receiver. Subscription bond from “Request()” function of the sender to “Receive()” function of the receiver (Figure 4.15) enables sender to make requests. Simultaneously the “Request()” function sends control to the “Receive()” function of the sender. This is enforced by having another subscription bond from “Request()” function to the “Receive()” of the sender with AND logic. Sender has to wait until it receives data from the receiver. This can easily be enforced by having a negotiation bond from “Receive()”

function of the sender to the “Reply()” function of the receiver. Negotiation bond makes sure that “Receive()” function keeps the control until it gets the reply from receiver. BPEL’s invoke/receive activities at senders site directly support this construct. Receiver’s site supports this construct using receive/reply construct.

4.3.2 Asynchronous Communication

Here, message sender continues its operation after completion of the message dispatch. It does not wait for the reply from the receiver. Synchronous communication also has three scenarios. *Message passing* is the simplest asynchronous communication method. Once sender makes the request it does not wait for the reply. Sender essentially forgets the request. Receiver processes the request. *Publish/subscribe* enables sender to determine the receiver based on the interest of the receiver. Then it dispatches messages only to the interested receivers. Finally, *Broadcast* can be seen as more relaxed version of publish/subscribe. Unlike publish subscribe when an event occurs it will be broadcast to all receivers regardless of their interest.

Publish/Subscribe: Publish/subscribe enables sender to determine the receiver based on the interest of the receiver. Receiver 1 has its interest in the event B which is identified by the function $F_b()$. This is enforced by having a subscription bond from $F_b()$ to $B()$ of receiver 1. Other two receivers, receiver 2 and receiver 3, have their interest in the event A which is identified by $F_a()$. This is enforced by having subscription bonds from $F_a()$ to $A()$ of receiver 2 and receiver 3. When an event $B()$ happens at sender 1, it will trigger the subscription bond from $B()$ of sender 1 to $F_b()$ of subscription list. $F_b()$ will in turn trigger appropriate subscription bonds. (In this case it is $F_b()$ to $B()$ of receiver 1).

Similarly when an event A() happens at sender 2, it will trigger the subscription bond from A() to Fa() subscription list. Fa() will in turn trigger the appropriate subscription bonds. (In this case, two bonds from Fa() to A() of receiver 2 and receiver 3)

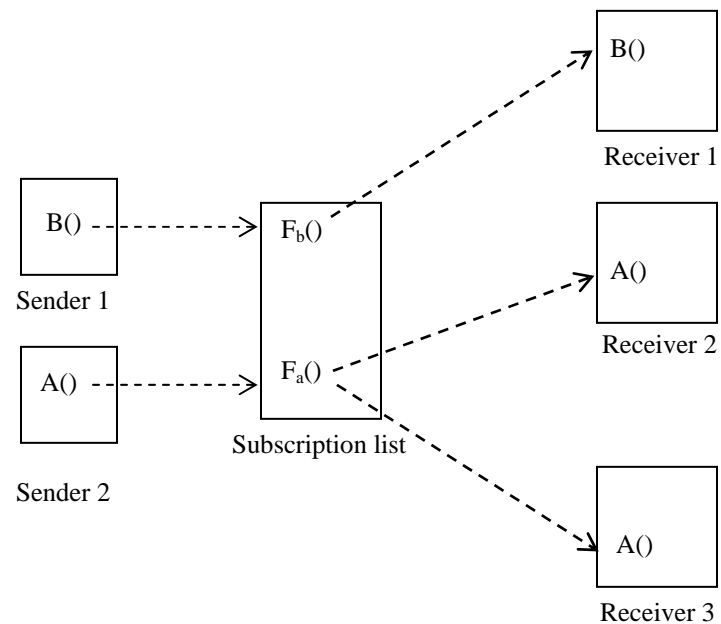


Figure 4.16: Publish-Subscribe Communication

Publish subscribe is not directly supported by BPEL. However, one can use BPEL's event handling functionalities to construct publish-subscribe scenario. But the designer has to put much effort designing the event handling mechanisms.

4.4 Related Work and Discussion

In this section we critically discuss languages and tools available for web service workflow coordination, modeling and expressive capabilities of these languages, and efforts towards formalizing web service coordination.

Many languages including WSFL [Ley01], WSCI [WSC102], WS-Coordination [WSC], WS-Conversation [WSCL], BPML [Ave02], XLANG [Tha01], BPSS [ebXML03], and BPEL4WS [Wee05] have emerged as WS composition languages [Aal03a]. However, these languages provide different techniques to compose web services without solid theoretical underpinning. Too many standards make the process complex and add ambiguity to the system [Hul04]. Some authors refer to these competing standards as the “web service acronym hell” [Aal03b]. Various research and standardizing efforts are underway to standardize web service composition technologies.

Interaction Pattern Based Analysis: Passing control and data among participant entities are carried out by establishing a communication channels among participants. Effective and efficient maintenance of the channel content is prime importance in SOC. Proper understanding about interaction patterns helps in this regard. In [Car99], authors have taken some initiatives towards such analysis. In [Aal03b], authors suggested that it is necessary to critically evaluate current coordination standards and develop unambiguous methodology to define web service coordination. In [Ben02], authors have taken a good initiative toward such framework by identifying various interaction patterns in web service composition. Research efforts such as [Lom01, Bic03, Zla03, Lim02] try to address the negotiation issues related to e-commerce. In [Ver05, McL02, Ko03, Kim02]

authors have identified problems and solutions to some of them related to negotiation process involved in supply chain management.

Modeling and representing negotiation logistics using formal tools such as Petri nets is important because such representation gives an opportunity to perform formal analysis. In [Hua02], authors discuss modeling e-negotiation activities using Petri nets. In that authors have pointed out that in e-negotiation among multiple agents. In [Rap00], authors have proposed a Petri net based model to manage interdependencies among collaborative tasks in workflows. In this scheme, workflow dependencies are mapped to coordination level by inserting adequate high-level Petri net models. HiworD [Ben03] is a Petri net based workflow design and simulation tool, which allows designers to model and simulate business process before deploying the actual workflow.

As we have discussed in section 3, in [Aal03c], authors have gathered a repository of workflow patterns that are common in workflow modeling and they have grouped them into six categories (Table 4.1). PhD thesis presented in [Kie02] has studied the expressiveness and suitability of workflow languages for modeling these control flow patterns. Also, in this thesis, Petri net has been used as the formal modeling tool. Such studies show that any workflow standard should have enough expressive power to model complex systems. Using those workflow patterns as a benchmark, web services composition and workflow languages such as BPEL4WS, XLANG, WSFL, BPML, WSCI, and High-level Petri-net-based languages have been evaluated [Wee05]. In[Woh03], authors have identified three good reasons to use Petri-net namely; a) Formal semantics, but easy to model graphical representations, b) State-based instead of just event based, and c) Abundance of analysis techniques. However, despite those

important properties Petri-net has difficulties dealing with complex workflow control patterns based on multiple instances, advance synchronization, cancellation [Aal02]. The difficulty lies due to the fact that Petri-net depends heavily on state-based rather than the event/message based. Due to distributed nature of today's information technologies (middleware, web services) underling techniques need to have both state as well as message handling capabilities [War 05]. BPEL on the other hand tries to satisfy these two requirements and is becoming popular among we services community as a workflow language. However, BPEL also has difficulties enforcing complex control flow patterns and the language itself is complex. This section discusses challenges Petri-net and BPEL face handling aforementioned workflow control flow patterns successfully. We note that an existing workflow modeling framework called "YAML" [Aal02] is also capable of handling all these control flow patterns. The difference between YAML and web bonds is that YAML has been specifically designed to enforce these control flow patterns (by essentially augmenting a Petri net based system) by adding explicit constructs for each control. In contrast, web bonds have been designed as a generic framework for coordination/collaboration among distributed systems and these happen to be capable of handling these workflow control flow patterns.

Theoretical Treatments of Web Service Coordination: In [Ben02], authors have pointed out that lack of fundamental primitives for web service integration has resulted in plethora of products and standards. These standards are overlapping, and are suitable for domain experts. They require refinement, consolidation, standardization and theoretical treatment to find a small yet powerful core set of threading primitives. In [Bru05], authors present a hierarchy of transactional calculi with increasing expressiveness. They

start from a very small language in which activities can only be composed sequentially. Then, progressively introduce parallel composition, nesting, programmable compensations and exception handling. In [Aal05], author discusses pros and cons of Petri nets and Pi calculus for web service conversion languages (WSCL) and illustrates fundamental differences between Petri nets and Pi calculus. A choreography language named CL [Bus05] is another noticeable effort towards formalizing web coordination. Following the approach of WS-CDL, in CL choreography contains a “global” definition of the common ordering conditions and constraints under which messages are exchanged within a conversation among collaborating services. In [Luc05], authors argue that three different mechanisms for error handling available in BPEL are not necessary in web service composition. They have formalized a novel orchestration language based on the idea of event notification as the unique error handling mechanism, and present a formal definition of three BPEL mechanisms in terms of their calculus. In [Coo05], authors propose a programming language which directly supports web service development, leverages XQuery for native XML processing, supports implicit message correlation and has high level calculus-style concurrency control. However, such developments are in very early stage and much remains to be done to find a web service “coordination theory.”

4.5 Summary

PhD dissertation presented in [Kie02] has studied the expressiveness and suitability of workflow languages for modeling these control flow patterns. Also, in this thesis, Petri net has been used as the formal modeling tool. Such studies show that any workflow standard should have enough expressive power to model complex systems. Using those workflow patterns as a benchmark, web services composition and workflow languages such as BPEL4WS, XLANG, WSFL, BPML, WSCI, and High-level Petri-net-based languages have been evaluated [Wee05]. In [Woh03], authors have identified three good reasons to use Petri-net namely; a) Formal semantics, but easy to model graphical representations, b) State-based instead of just event based, and c) Abundance of analysis techniques. However, despite those important properties Petri-net has difficulties dealing with complex workflow control patterns based on multiple instances, advance synchronization, cancellation [Aal02]. The difficulty lies due to the fact that Petri-net depends heavily on state-based rather than the event/message based. Due to distributed nature of today's information technologies (middleware, web services) underlying techniques need to have both state as well as message handling capabilities [Wee05]. BPEL on the other hand tries to satisfy these two requirements and is becoming popular among web services community as a workflow language. However, BPEL also has difficulties enforcing complex control flow patterns and the language itself is complex. This section discusses challenges Petri-net and BPEL face handling aforementioned workflow control flow patterns successfully. We note that an existing workflow modeling framework called "YAML" [Aal03] is also capable of handling all these control flow

patterns. The difference between YAML and web bonds is that YAML has been specifically designed to enforce these control flow patterns (by essentially augmenting a Petri net based system) by adding explicit constructs for each control. In contrast, web bonds have been designed as a generic framework for coordination/collaboration among distributed systems and these happen to be capable of handling these workflow control flow patterns. Moreover, web bonds are capable of modeling all the benchmark workflow control flow patterns and distributed communication patterns.

CHAPTER 5

WEB COORDINATION MANAGEMENT MIDDLEWARE SYSTEM

As we have mentioned in Chapter 1, existing workflow technologies over web services are constrained by the stateless architecture of the web services. This typically results in complex and centralized logic for workflow coordination. Coordination technologies such as web coordination bonds enable distributed coordination. However, currently web services are not capable of maintaining and managing coordination and enforcing their own dependencies. Key architectural enhancements are needed to transform the stateless web services into state-preserving self-coordinating entities to allow distributed coordination. Such capability enhancements in the web services will also lead to simpler coordination logic. In this Chapter we present our Web Service Coordination Management Middleware (WSCMM) that is a simple but powerful enhancement to the web service infrastructure enabling the services locally manage the dependencies and the handle messages resulting from multiple workflows. The development of a WSCMM is analogous to the development of a DBMS (database management system) to coordinate the execution of queries and transactions in the web services domain.

We have carried out a detailed simulation to identify and key components and design issues of our middleware. Also, we compare and contrast our architecture with the current web service technologies, and present details of a prototype implementation. Proof-of-concept experiments demonstrate that we can develop both centralized and distributed workflows over the architecturally enhanced web services with relative simplicity.

Chapter 7 presents simulation details and Chapter 8 discussed the prototype implementation details. Rest of this Chapter has been organized as follows. First, we revisit the current state of the art in web service workflow development and present our vision. Also, pinpoint issues pertaining to current web service based workflow development and deployment. Then, we propose our middleware solution and identify key components and their functionality.

5.1 Limitations of Current Centralized Coordination

Service composition is the process of aggregating standalone (Web) services together to form another value-added service based upon pre-defined application logic. Usually, composed service is state preserving and acts as the central coordinating agent. The constituent services can be from different organizations providing way to develop inter-organizational collaborative applications (Figure 5.1).

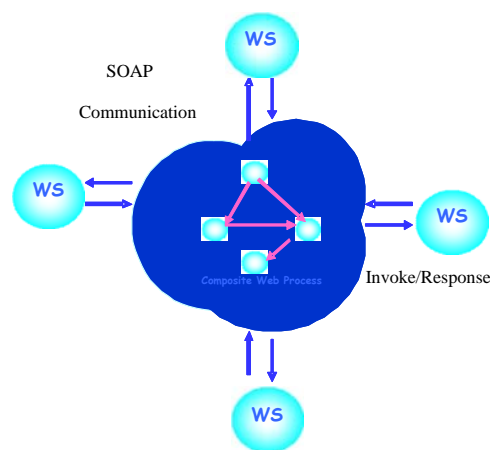


Figure 5.1: Current State of the Art: Composite Web Process as a Central Coordinator

Due to its centralized nature and the inability of participant web services to share the burden of enforcing composition and coordination constraints, composed web process

has to encapsulate numerous functionalities ranging from application logic to transaction management. There are two distinct sets of problems of this model.

Detailed level programming: A composed web process needs to encapsulate numerous functionalities ranging from application logic to transaction management. It is the designer's responsibility to focus on low level (atomic) details such as message correlation, and state (context) information to high-level application logic. Therefore, current technologies such as BPEL are at the level of the assembly language for web service composition and coordination.

Centralized coordination: Due to the current architecture of the composed web process it becomes a central coordinating agent. There are both pros and cons in centralized coordination; the positive point is being total control over the behavior of the web process. However, distributed coordination has two categories of advantages over centralized coordination: (i) Due to security, privacy, or licensing imperatives, some web-based objects will only allow direct pair-wise interactions without any coordinating third-party entity; and (ii) Centralized coordination/workflows suffer from issues such as scalability, performance, and fault tolerance [Gir04]. For example, data transfer and message passing among participant web services need to go through the central web process generating more network traffic and making the composed web process more complex. Efforts such as IBM symphony [Gir04] try to eliminate centralized coordination by partitioning centralized BPEL code into separate modules so that they can run in a distributed setting. However, there are limitations to such efforts. First, it is necessary to develop the centralized BPEL code and then distribute it. Second, usually, there are

problems partitioning the code in complex application scenarios such as long running transactional applications without proper infrastructure support.

Solution: In order to overcome above limitations, it is necessary to: i) Extract higher-level abstractions such as coordination and message correlation, which are independent from the application logic of the composition, and ii) Distribute these responsibilities among constituent web entities. This will transform the web services we know today into conversation and coordination aware stateful web entities and make the application development less intensive [Jor05, Bar05, Sch05, Bou05, Wan05, Tai04]. We envision web service actively participate in workflow enforcing their own dependencies as shown in Figure 5.1

Chapter 7 discusses relevant important developments on web service composition, coordination, and enhancements to the basic web service infrastructure

5.2 Evolution of Database Application Development

A good motivating analogy would be to consider the evolution of database application development platforms.

Fig. 3 illustrates the evolution of database technologies from simple file system to a three-tier system, equipped with layers to manage the database, user interface, and workflows, progressively reducing the burden of application development. In early 60's, application developer had the burden of capturing all the logic of data manipulation, constraint checking and concurrency control (Figure 5.2a). With the introduction of database

management systems (DMBS), most of the data handling functionalities was transferred to DBMSs. Development of various middleware technologies and workflow management systems further reduced the burden of application developer (Figure 5.2d).

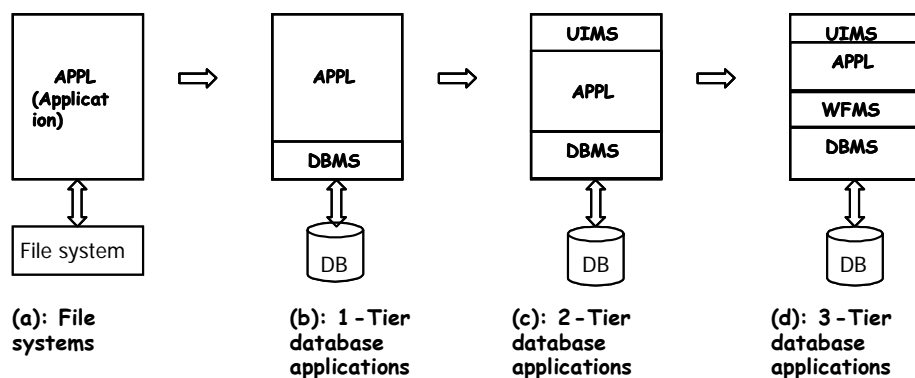


Figure 5.2: Evolution of Database Application Infrastructure [Aal98]

The current, web server based applications and the first stage database applications have similar characteristics. Application programmer has the burden of capturing all the application logic as well as house keeping tasks. The individual Web services, encapsulating information and data stores, with its access methods described using Web Service Description Language (WSDL), lacks even the basic management system (Figure 5.3a), not to mention any support for transactions, composition, or workflows. Application programmer has the burden of capturing almost all of the coordination logic. From this perspective, Web services infrastructure is still in its early developmental stage.

Therefore, we propose to a) Enhance the web services infrastructure so that it has a management system for web services to manage methods and method invocations more effectively akin to DBMS in databases, b) Evaluate coordination and composition techniques for Web services and transfer generic functional layers to Web service side so

that they become capable entities to enforce distributed coordination akin to WFMS in databases. We call them Web Service Management System (WSMS) and Web Service Coordination Management System (WSCMS) respectively. The following section presents our architecture.

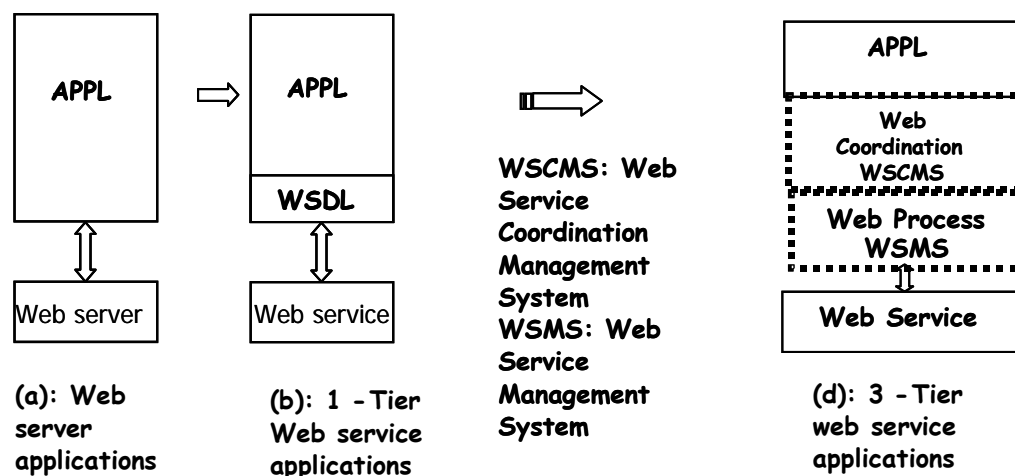


Figure 5.3: Proposed development for web service infrastructure

5.3 Functionalities Encapsulated by the Centralized Workflow

Here, we identify functionalities encapsulated by the composite web process and requirements for distributed web service coordination. Then we layout requirements of a middleware system for distributed workflow coordination over web services.

Requirements: The composed web process needs to encapsulate numerous functionalities ranging from application logic to transaction management. Following major

functionalities are being encapsulated by the composed web process to implement such requirements [Alo04, Jor05, Pra05, Bar05, Ver05].

1. Modeling execution control (internal coordination): Integrate autonomous web services together to encapsulate the application logic. In literature this also is referred as the abstract process [Bar05].

2. Modeling external coordination among constituent web services: Enforcing dependencies and constraints among participating web services. This entails ensuring proper communication context, representing the role of each participant and reliable messaging. This also requires proper sequencing of messages and correlation.

3. Remote service invocation: In SOA, services expose services available as public available methods so that requesters can invoke and get the service done. Theoretically, the concept is the same as Java RMI or CORBA remote method invocation. However, the difference is that services are autonomous entities and service requesters do not have details of service implementations.

4. Context information handling: Long running collaborative applications need context (state) information to be stored and processed.

5. Event handling: Web service communication is message based and events are notified using messages. Event notification may imply an invocation (triggering) of some functionality.

6. Transaction support: Inter-organizational collaborative applications may have some transactional context. Such applications need to ensure rigid or relaxed ACID properties. Moreover, they need to support compensation and error handling.

Based on above functionalities we extract three key layers of functionality encapsulated by the composite web process (Figure 5.4). Top layer encapsulates the abstract workflow process defined using high-level constructs. Middle layer represents the code that enforces workflow dependencies (implements based on underline language constructs). Last layer implements actual communication with individual web services that are participants of the workflow. For each workflow, all three layers need to be implemented from the stretch. However, 2nd layer and 3rd layers represent significant amount of generic functionalities such as enforcing basic workflow coordination logic, Web service invocations, message handling and storing corresponding state information. Therefore, generic functionalities of these two layers can be extracted and provide as a middleware layer for distributed workflow coordination. We identify following three categories of functionalities for a middleware system for distributed workflow coordination over web services.

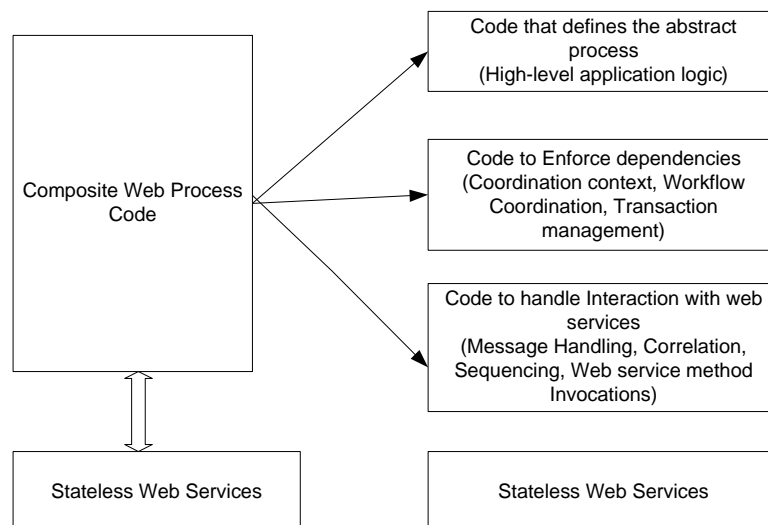


Figure 5.4: Functional decomposition of composite web process

Implications on Functionality:

1. Enforce dependencies: Workflow activities need to satisfy various kinds of constraints in order to accomplish the task successfully. For example, before initiating the activity, it may need to satisfy application specific data, control and resource dependencies and once activity is completed activity may need to inform results and pass control to other activities if the workflow based on various conditions. In a distributed coordination environment, each web services needs to maintain its own dependencies and enforce them locally.

2. Preserve state information: Long-lived workflow applications require state of method invocations (success or failure) and intermediate results to be stored and make global decisions. Such state information needs to be maintained and correlated with proper application context.

3. Process messages: Web services communicate exchanging messages. Therefore, in order to become live participants in distributed applications, web services should bear enough capabilities to process messages and make decision accordingly. This entails maintaining proper communication context for each application, message correlation and sequencing, and reliable messaging.

In our middleware, functionalities pertaining to workflow dependency are carried out by WSCMS layer. Processing messages and maintaining state information is handled by WSMS. Next section discusses these components in detail.

5.4 Web Service Coordination Management Middleware Architecture: An Overview

This section starts with a generic description of our web WSCMM architecture, its components and related issues. Then we discuss each component in detail. The web coordination middleware consists of two main components: Web service management system (WSMS) and the Web service coordination management system (WSCMS). Note that our middleware clearly distributes the workflow among three distinct functional layers (Figure 5.5). These two components are attached to the service provider, i.e. a layer between the SOAP (any other communication) and WSDL enhancing the internal architecture of web services (Figure 5) [Alo04].

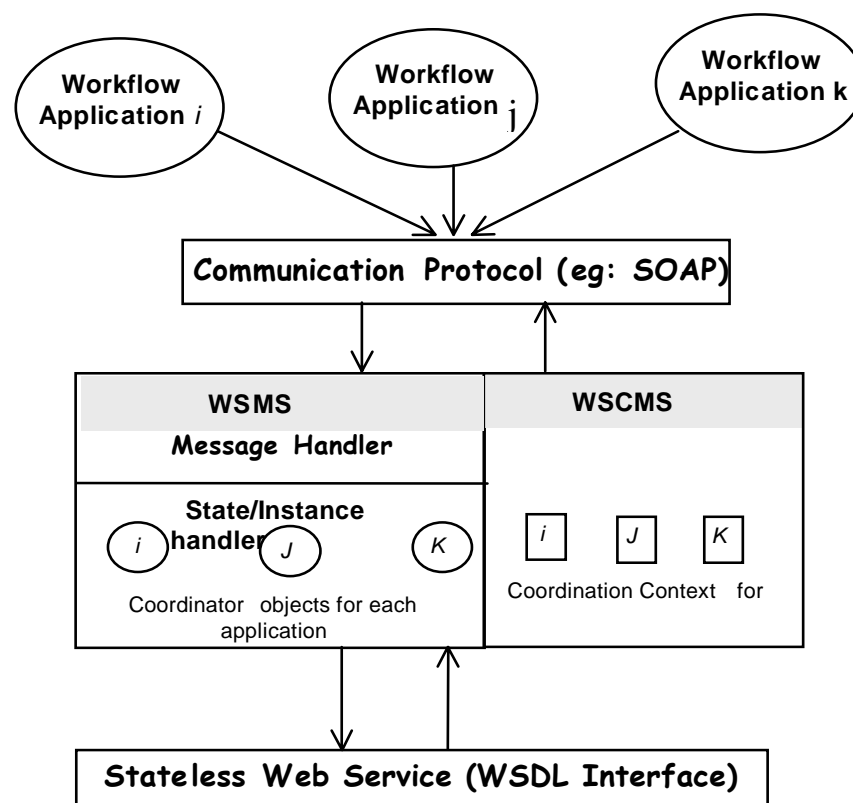


Figure 5.5: Web Service Coordination Middleware Overview

Web Service Management System (WSMS): WSMS handles two functionalities; Preserve state information for long-live interactions and process messages locally and initiate appropriate actions.

Stateful view: State/instance handler instantiate a coordinator object based on WSDL description for each such application. Coordinator object has a binding to the original web service method calls. Moreover, each coordinator object has a corresponding status context stored in the persistent storage. WS method invocations go through the coordinator object. Each method invocation has series of steps including enforcing dependencies and updating state information.

Message handling: Message handler of the WSMS handles the inter-web service communication and keeps the state information of interactions. Upon an arrival of a message, communication server (SOAP server) passes it to the message handler. Message header conations a unique identification for each message (ConvID). ConvID consists of a reference to the application, method being invoked, parameter set, status of tag of the invocation such as "Ready", "Commit" in transaction processing. Based on this information, message handler resolves the message and takes appropriate actions.

Web Service Coordination Management System (WSCMS): Keeps the coordination (dependency) information (coordination context) for each application and enforces dependencies. Since coordination and dependency enforcement is local to each participating web service, WSCMS maintains coordination context for each applications locally to reflect dependencies. Web services coordination management system supports two types of dependencies: pre method execution dependencies and post method execution dependencies. In addition it supports two types of long-lived interactions:

transaction-oriented and non-transaction oriented coordination. Transaction oriented coordination requires participants to perform some sort of a commit processing while non-transaction oriented coordination requires only all dependencies to be fulfilled before and after the execution of a particular method global or group decision may not be needed.

5.5 Web Service Coordination Management System

In web service based workflow applications, individual web service represents a particular workflow activity. Activity performs its operation by invoking web service method calls. Workflow dependencies need to be associated with WS method invocations. Typically, workflow activities enforce two types of dependencies. Before initiating the activity (triggered by the workflow engine) it needs to make sure that all the dependencies (including data, control and resource) have been satisfied. If not, activity waits until it receives all the control and data items or it can start fulfilling these requirements. These kinds of dependencies can be characterized as “pre execution dependencies. “ Other type of dependency arises once workflow activity is completed. Upon completion of the activity, it may require to pass control/data to other entities in the workflow based on workflow specific constraints. These kinds of dependencies are characterized as “post execution dependencies.”

Pre Execution Dependencies (join dependencies): Pre execution dependency for workflow j , defined over the method m_i of web service w_i with parameter set k can be represented as $J_{j.w_i.m_i}(\text{param}_k) = \{D, \text{constraints}\}$, where D is the set of destination methods, and constraints are workflow constraints such as AND-join and Sync-Merge.

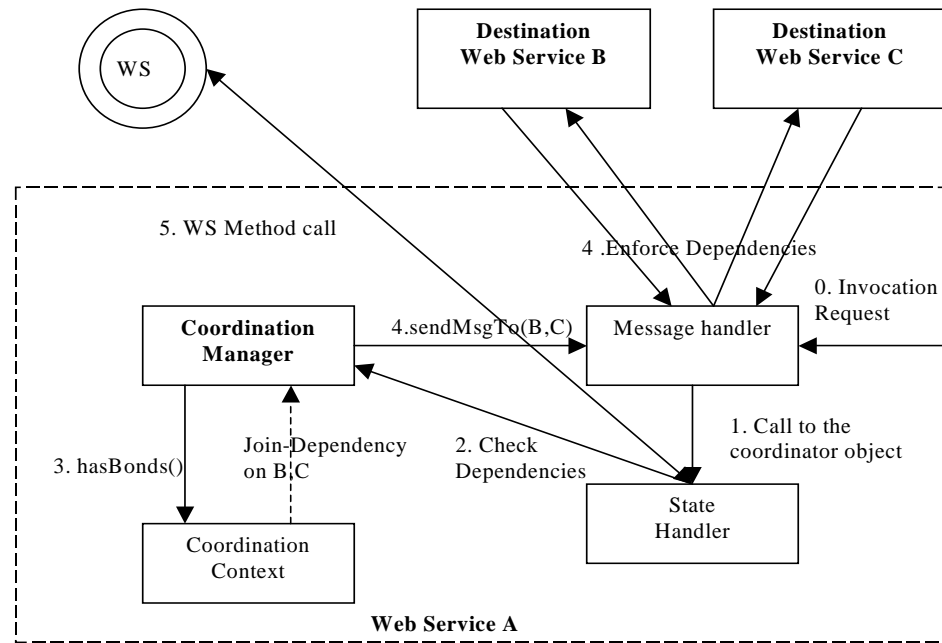


Figure 5.6: Enforcing Pre Execution Dependencies

WSCMS ensures that join-dependencies are met before making the web service method call. Series of events take place in local WSCMS as well as destination WS's coordination management systems while enforcing join-dependencies. Figure 5.6 illustrates the interaction among WSCMM components while enforcing join-dependency constraints. Message handlers maintain an inbox and outbox for each workflow application. Both inbox and outbox has entries for each join-dependency point. When it receives control/data from destination entities message handler direct them for the appropriate

inbox. Once the activity receives trigger (control) to perform the method call (step 0), it sends a message to the WSCMS for dependency check (step 1, 2, and 3). If all the dependencies are met web service method get invoked and state information is updated (step 5). Otherwise, WSCMS sends messages to all the remaining destination entities for dependency check (step 4). Dependency check performs two operations. First, it request states information from the state handler of the destination web service related to this particular application join-point. If status information is available respond is sent. Otherwise, it tries to invoke the remote method and send the response to the requester web service. This invocation requires similar dependency check.

Post Execution Dependencies (split dependency): Split dependency for workflow j , method m_i of web service w_i with parameter set k can be represented as $S_j.w_i.m_i(\text{param}_k) = \{D, \text{constraints}\}$, where D is the set of destination methods and constraints are workflow constraints such as AND-Split, XOR Split.

Enforcing split dependencies require web service to trigger set of remote web services depending on the workflow constraints specifies for the split-point. Figure 5.7 illustrates the interaction among WSCMM components while enforcing split-dependency constraints. As shown in Figure 5.7, WSCMS requests the message handler to send data/control to remote web services according the workflow split criteria. Message handler places remote invocations to the outbox (dispatcher) and triggers remote web service methods (step 6). At the same time coordination management system updates state information (step 5).

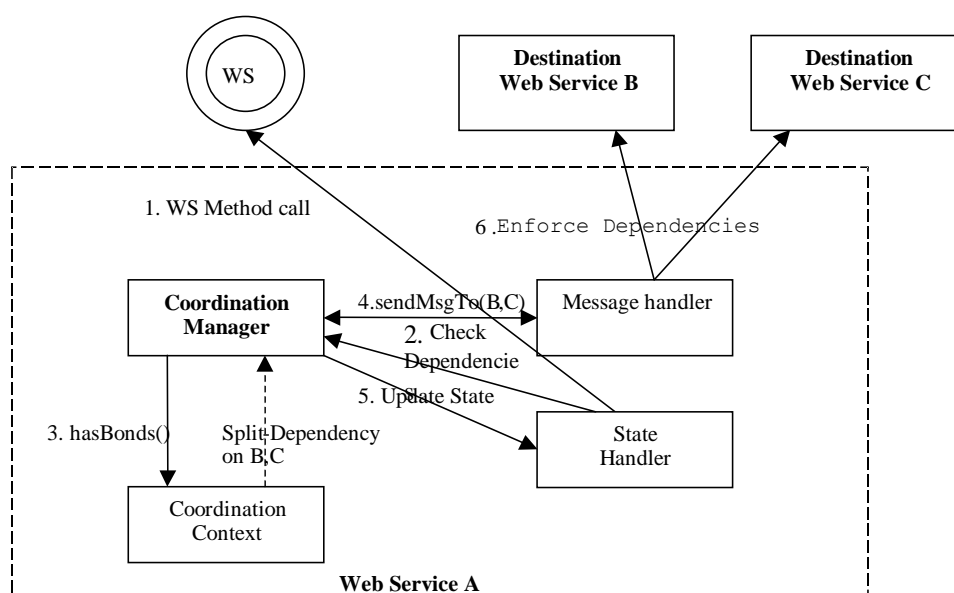


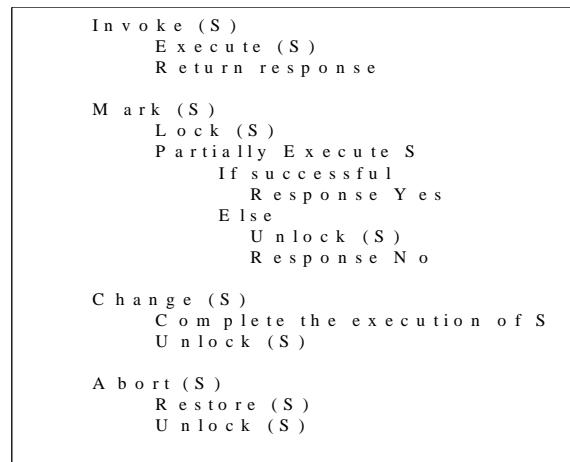
Figure 5.7: Enforcing Post Execution Dependencies

To enforce join-dependencies and split-dependencies WSCMS support two types of coordination mechanisms: transaction-oriented coordination and non-transactional oriented coordination.

Transaction-oriented Coordination: In order to support transactional behavior i.e., an ability to partially execute the triggered method and go to “ready to commit” state, which can either be committed or aborted subsequently by the triggering entity. Alternatively, a reservation/locking facility on methods with specified parameters (thus indirectly reserving certain changes on specific data components) are needed. In order to support such behaviors, WSCMS should provide two kinds of method invocations. Assume a method/service S at a Web Service. One can normally execute method S ($invoke(S)$) or, to support the dependency behavior of a transactions, partially execute S and reserve/lock it

(*Mark(S)*) and, subsequently based on group decision, complete execution of *S* (*Change(S)*) or abort its execution (*Abort(S)*). The generic semantics operations described below may be implemented in various ways.

Semantics: (may not be implemented this way)



This kind of behavior is required in DBMS transaction manager.

Non-Transaction-based Coordination: For non-transactional coordination WSCMS needs to have the capability to trigger methods in other Web services and enforce simple data and control dependencies. Most of the split-dependency enforcements require non-transactional behavior. Consider that S_1 , S_2 , and S_3 are different web service methods. After executing S_1 , S_2 and S_3 need be executed (control/data dependency, S_1 triggers S_2 and S_3). Functionality for simple trigger can be described as follows

Semantics (may not be implemented this way): Mark S_1 ; If successful Change S_1 then Try: Change S_2 and Change S_3 . Note that the "try" may not succeed. And there may be timeout mechanisms to avoid deadlocks.

5.6 Web Service Management System

WSMS consists of a message handler, state/instance manager and application context manager (Figure 5.8). The core functionality of the web service management system is to transform the stateless web service into a state preserving self-coordinating entity. WSMS performs this transformation by generating a coordinator object to represent the web service, which encompasses all the coordination capabilities of the underline WSCMS implementation. Figure 5.9 illustrates the architecture of the coordinator object. The coordinator object provides the same interface as the web service provides to the outer world. Web service method invocations of the workflow take place through the coordinator object and the web bond coordination layer ensures that pre and post method invocation dependencies are satisfied. This indirection allows us to bring transparency to the system and hide the necessary coordination and communication logic behind it. As shown in Figure 5.10, each web service method call is encapsulated by join-dependency and split-dependency check. This logic ensures that workflow dependencies are satisfied with associated WS invocation.

Message handler	State/instance handler (Maintains live communication with the application)
	Workflow Context Manager (Maintains runtime time information per workflow application basis)

Figure 5.8: Web service management System

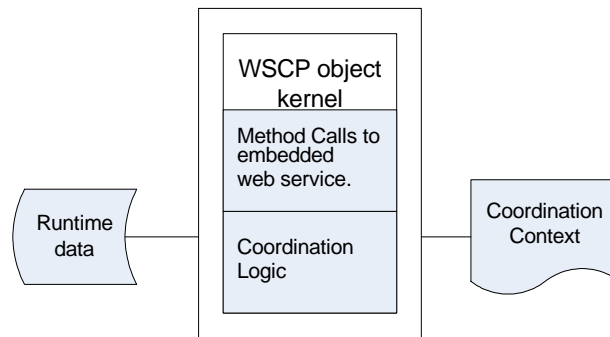


Figure 5.9: Coordinator Proxy Object Architecture

The idea of Web service coordinator proxy object together with underline bonding (workflow dependency modeling) primitives encapsulates the workflow coordination layer. This simple, but powerful idea empowers web services and makes workflow configuration less programming intensive. We believe this concept carries enough potential to lead a fundamental shift in workflow development over web services. Instance/state handler implements coordinator object functionalities.

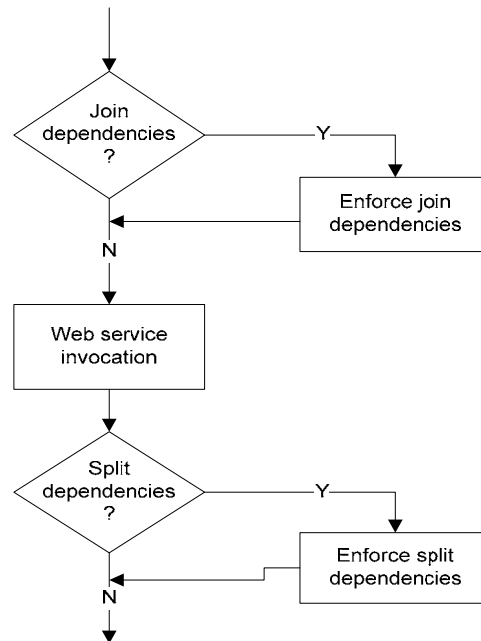


Figure 5.10: Typical Flow within a coordinator proxy object

Status and Status Information: State information reflects the current the snapshot of method invocations (success or failure). State information is stored in a persistent storage. This is required for long-lived interactions (duration of such interactions can be several minutes to few weeks). Instantiated coordinator objects have a unique identifier and can run few minutes to several weeks/months. They can be accessed asynchronously (required by long lived transactions). The state information stored in the persistence storage includes method invocation details (e.g: transactional oriented coordination) and intermediate results.

Workflow Context Manager: Workflow context manager allows multiple workflows to be defined over same web service concurrently. Application context manager stores state information based on the application ID. Each workflow in is assigned a unique ID.

State/Instance handler assigns a unique ID for each application and associates it with application state, coordinator proxy object, and the coordination context of the application. Each message is associated with this unique ID and the message handler uses this ID together with other invocation related information to handle message correlation.

Message Handler: Message handler receives method invocation and other workflow related messages (data, control, and triggers). Message handler keeps separate communication contexts for each application. Communication context consists of two parts: Inbox and Outbox. Inbox is the placeholder for incoming requests and out put is the placeholder for out going messages. Upon receipt of the message, message handler determine appropriate message box and take appropriate action. This architecture enables message handler to perform message correlation and message sequencing.

Message Correlation and Sequencing: The conversation controller handles message correlation and message sequencing. Message correlation is the process of coordinating first invocation and subsequent invocations to the same web service method(s) in the context of some application scenario. In order to do this each message is augmented with a unique conversation id (ConvID) and the requester's method name and the parameter set (*<MessageContext:A:I:M_i:Covnid:Tag: RequesterMethodName >*). This information is passed to each coordinating entity with the message. Also, due to the network delays and the distributed nature of the application execution environment web service may receive messages in the different order compared to the order of invocation. It is the responsibility of the message controller to direct them to the proper inbox regardless of their arrival

sequence. This resolution can be done using the ConvID and method names (requester/supplier).

5.7 Summary

In this chapter we argued that web services infrastructure need to be enhanced for effective distributed coordination over web objects including web services. Towards this goal we presented the WSCMM architecture, a simple but powerful enhancements to the current web services infrastructure that transform passive stateless web services we know today into conversation aware, stateful web objects. Key to this transformation is the introduction of coordinator proxy object that lively participates in the workflow on behalf of the web service. Coordinator proxy object is stateful and is capable of enforcing and maintaining workflow dependencies. Chapter 6 presents simulation details and a comparison of our middleware with other similar architectures Chapter 7 discussed the prototype implementation details.

CHAPTER 6

SIMULATION AND VERIFICATION OF WEB SERVICE COORDINATION MANAGEMENT MIDDLEWARE

In chapter 5, we have discussed our Web Service Coordination Management Middleware (WSCMM) in detail. The primary objective of the WSCMM system is to distribute the workflow coordination responsibilities among participating web services. Subsequently, it simplifies the workflow development process. As we have illustrated in the previous Chapter, WSCMM consists of two components: Web Service Management System (WSMS) and Web Service Coordination Management System (WSCMM). WSCMS maintains and enforces workflow dependencies while WSMS transforms the stateless web service into a stateful entity through the coordinator proxy object. Web service method invocations go through this object, which enforces pre and post web service method invocation dependencies using the functionality of WSCMS.

In our system, we have employed web coordination bonds to model dependencies. Therefore, WSCMS essentially maintains web coordination bonds and manipulate them in order to enforce workflow dependencies. Dependencies are stored in a “Bond Repository”. Bond repository is a persistent storage where each workflow has a corresponding bond store. Modeling various workflow and other dependencies using web coordination bonds have been presented in chapter 3 and chapter 4.

In this chapter first we discuss the realization of WSCMM using web coordination bonds. Then, we define the simulation model to verify the correctness of our architecture. We believe that WSCMM is a generic architecture and does not tight to any technology.

Thus, we discuss a possible realization of our middleware using other web service standards. Finally, we compare and contrast our approach with other similar efforts.

6.1 Realization of WSCMM Using Web Coordination Bonds

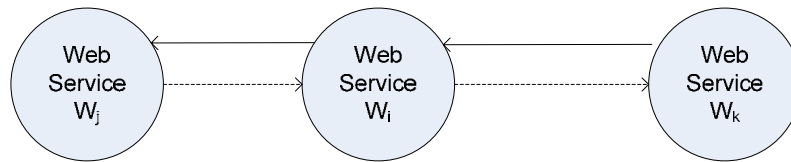


Figure 6.1: Enforcing Dependencies Using Web Coordination Bonds

Consider a situation where web service W_i , W_j and W_k participate in a workflow and the execution sequence is W_j , W_i and W_k respectively. In this case, before executing the appropriate method in web service W_i , it has to make sure that W_j has already being executed. Also, it needs to receive control/data from W_j . Then, W_i has to make sure that it passes required data and control to W_k after the execution. First two dependencies represent pre execution dependency and third one represents the post execution dependency for W_i . Having a negotiation bond from W_i to W_j and a subscription bond from W_j to W_i enforce the first dependency. Having a subscription bond from W_i to W_k enforces the second dependency.

When we model and execute such dependencies using our middleware platform. It is the responsibility of the coordination management system of each web service to store these bonds and enforce them. Messages are being sent and received using web bonds to enforce these dependencies. Therefore, the message handler of the web service management system should be capable of receiving messages from bonds, resolving them, and directing them to appropriate components for further processing.

The message handler receives messages from subscription bonds with data/control or for method invocations. It also receives messages from negotiation bonds to enforce pre execution dependencies (eg: W_i to W_j). Once a web service receives these messages it resolves the message and takes appropriate actions. Components of the middleware interact internally during this process. Table 6.1 summarizes the external messages when modeling dependencies using web coordination bonds. Each message has a tag, and the message tag indicates the purpose of the message.

Table 6.1: External Messages among Web Services When Enforcing Dependencies Using Web Coordination Bonds

Message Type	Source	Tag
Incoming	Method Invocations from Remote web services (SB)	0
	Data/control from remote subscription bonds (SB)	1
	Method Invocations (enforce dependencies) from Remote web services (NB)	2
Outgoing	Method Invocations to remote web services	0
	Data/control to remote web services	1
	Method Invocations (enforce dependencies) to Remote web services (NB)	2

We simulate the following scenario (Figure 6.2) to verify our architecture. First, negotiation bond based pre execution dependencies have been simulated. Then, subscription bond based post execution dependencies are modeled. We have used Discrete Event System Specification (DEVS) model tool. In the next section we describe the DEVS environment briefly.

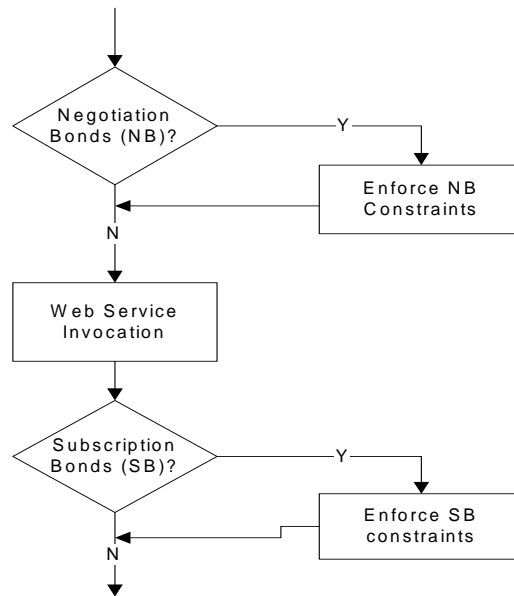


Figure 6.2: Simulation Scenario

6.2 Background: Discrete Event System Specification (DEVS)

Discrete Event System Specification (DEVS) provides formal framework that facilitates simulation and verification of distributed systems. DEVS is derived from mathematical dynamical system theory [DEVSJava]. It supports hierarchical modular composition and object oriented implementation. There are two primary modules: atomic model and coupled model. One can combine these models to specify complex simulations. Figure 6.3 shows the hierarchical modular composition of DEVS system.

DEVS Hierarchical Modular Composition

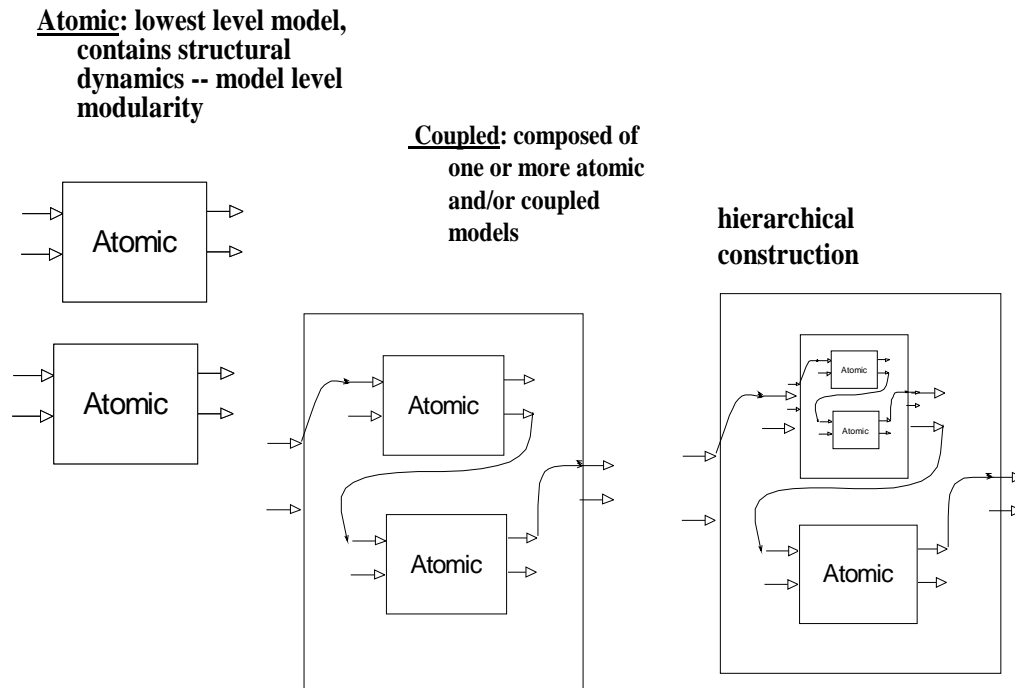


Figure 6.3: DEVS simulation model

Atomic models have input events, output events, state variables, state transition functions, external transition, internal transition, time advance function, computing function, and transitions. Current state can be specified using state variables and input and output functions are computed based on the current state and the computing function. Coupled model has components, interconnections, internal couplings, external input couplings, and external output couplings.

6.3 Simulating WSCMM Architecture

The main purpose of the simulation is to verify the correctness of our middleware and to identify design issues. In order to do that, we simulate the interactions among components of the middleware for different incoming messages including pre and post method invocation dependencies. We also simulate a simple sequential workflow and verify the correctness of our architecture.

Figure 6.4 shows our simulation model for the middleware. It consists of three main modules: message handler (msgHandler), web service coordination management system wsCoMys), and web service management system (wsMgtSys). Here, we briefly describe each component of the simulation model. Then we present the simulation results for following four scenarios for the correctness of our architecture. In particular, we illustrates that the web bond based realization of the WSCMM behaves correctly while enforcing workflow control flow dependencies.

Simulation Scenarios

1. Enforcing workflow dependencies using subscription bonds (post conditions)
2. Enforcing workflow dependencies using negotiation bonds (precondition)

6.3.1 Message Handler

The message handler consists of three components, two incoming ports to receive messages and three outgoing ports to send messages. Message receiver (mrec), receives messages from remote services (Figure 6.4). Upon receipt of the message, it places the message in a FIFO queue. Then, mrec passes messages to the message revolver (mres). Message revolver's job is to identify the type of message (Table 6.1). Based on the

message type, the message is directed to the appropriate component. For our simulation, we have used the following format of the message.

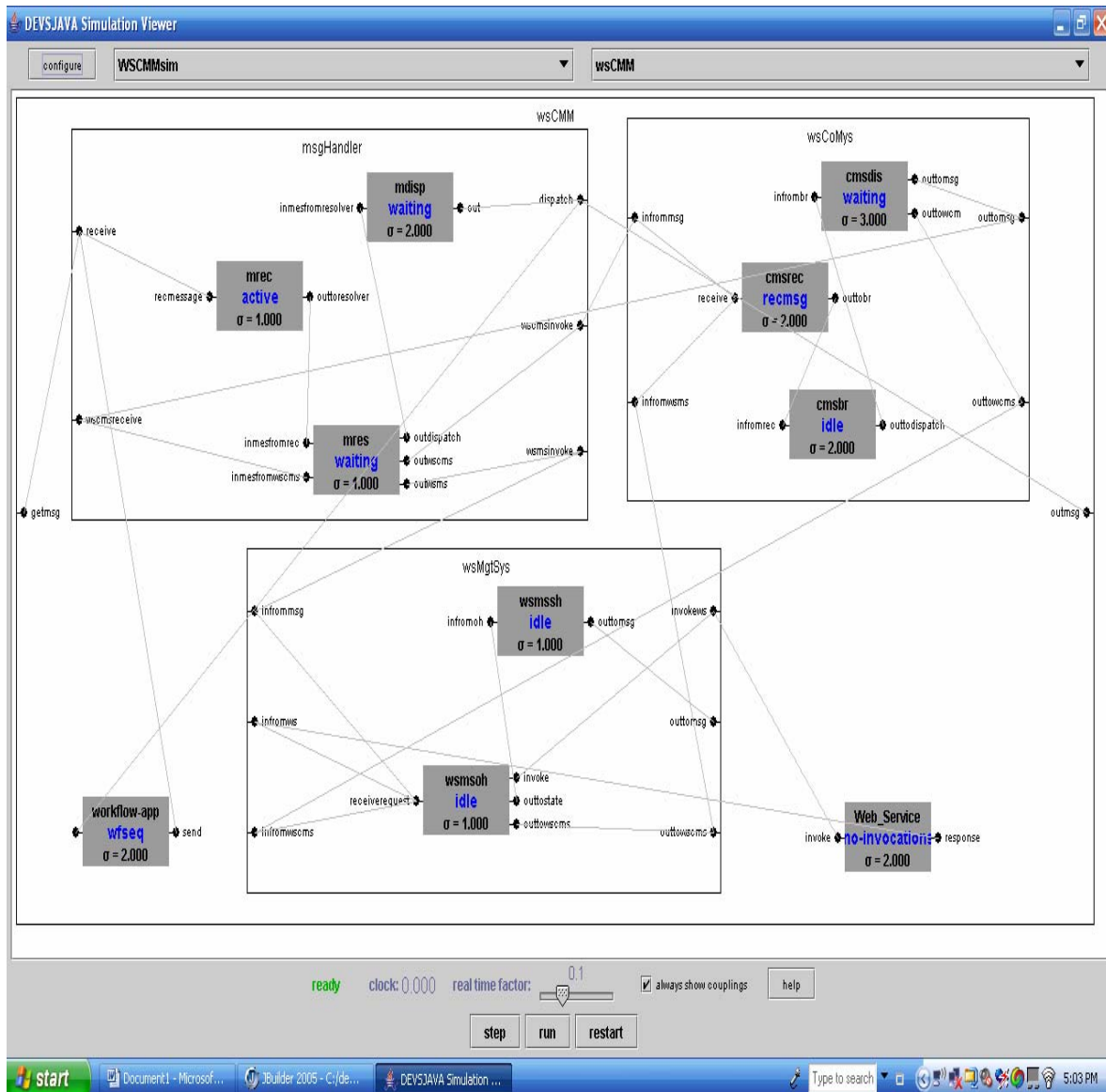


Figure 6.4 : WSCMM Simulation Model

Message format

Workflowid:fromwebservice:method:parameterset:tag

The first portion is to identify the workflow because any web service can participate in different workflows at a given time. Second portion is to identify the message sender. Third and forth portions contain method details and parameters. Finally, the tag is to identify the type of message. For example, suppose web service w1 receives the message, wf1:ws2:m2:p2:0. This means that the message belongs to workflow 1. Sender is web service 2 and the tag is 0. Tag 0 means the message is a method invocation. In this case, invocation of method m2 with parameter set p2. Once, the resolver receives this type of message it resolves the message using the tag and direct it to the appropriate output port. Table 6.2 shows the relationship between tag and the outgoing message port.

Table 6.2: Message tag and the outgoing message ports at the Message Handler

Tag	Outgoing port
0 -Method invocation	Send the message to wsms through “outwsms” port.
1-data/control from subscription bonds	Send the message to wscms through “outwscms” port.
2-enforce dependency (method invocation), negotiation bond	Send the message to wsms through “outwsms” port.
6-Enforce post method execution dependencies (data/control through outgoing subscription bonds)	Send the message to dispatcher through “outdispatcher” port.

Other possible functionalities of the message receiver of the message handler are checking appropriate security and enforcing QoS requirements. We have not considered them in our implementation. This simulation can be extended to accommodate such situations. Dispatcher of the message handler sends outgoing messages to remote web services. We

have modeled it using FIFO queue. However, the efficiency of this can be improved using multi-threaded dispatcher.

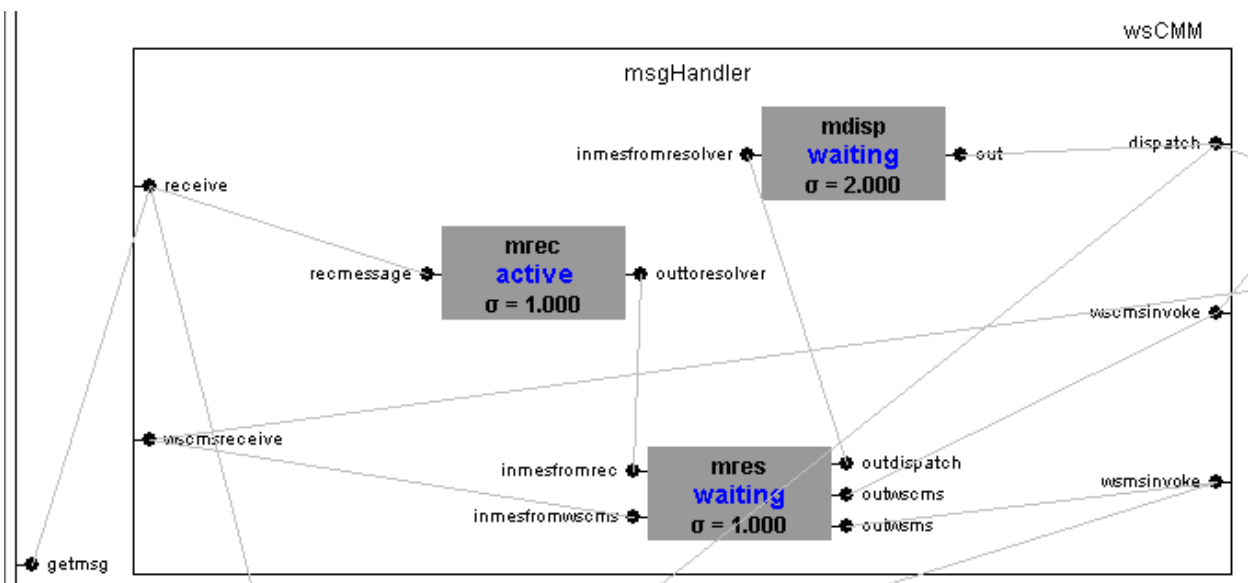


Figure 6.5: Message Handler

6.3.2 The Web Service Management System

The web service management system receives messages from three ports: *infrommsg*, *infromws*, and *infromwscms* (Figure 6.6). First, WSMS receives method invocation (tag 0 or 2) messages from the message handler. Then, it identifies proper web service through *websericeid* tag of the message. Upon identification of the workflow, it sends the message to web service coordination management system to check/enforce pre workflow execution dependencies. If dependencies are successfully met, then WSCMS changes the tag of the message from 0 or 2 to 5 and sends back to WSMS. Upon receipt of a message with tag 5, WSMS (*wsmsoh*), invokes the web service method. Web service invokes method and sends the results back to WSMS.

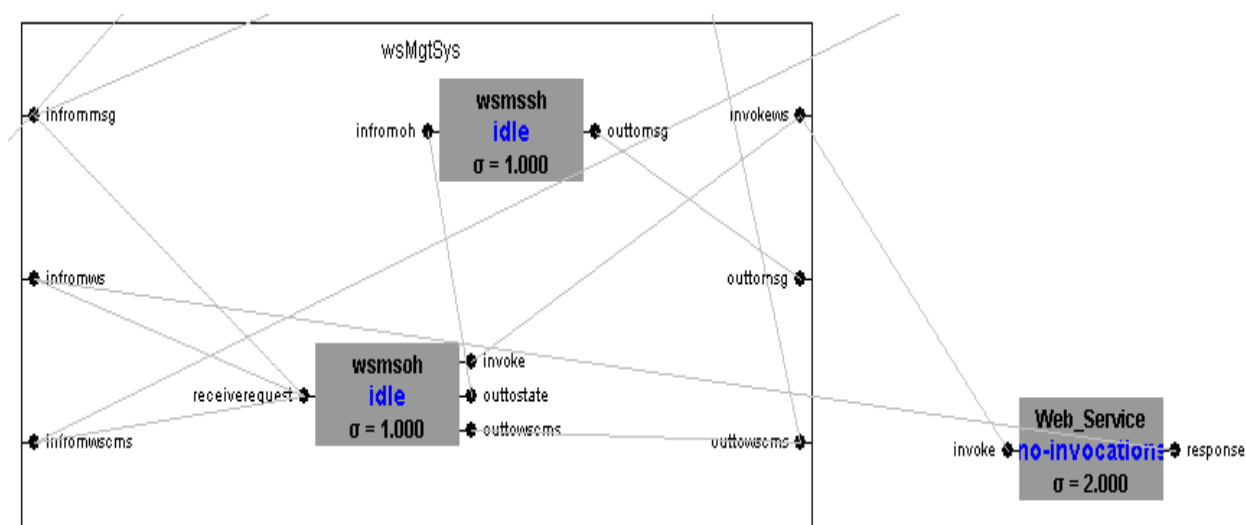


Figure 6.6: Web Service Management System

This time tag is 6. Tag 6 indicates that the method invocation happens (success or fail) and it need to update the state information with partial date or failure message. This is done by passing this data to the wsmssh (state handler). It stores these data in a file. In our simulation, this operation has been simulated by accessing a file having the same name as the workflow. Unavailability of such a file indicates an error. It also needs to send a message to the WSCMS to enforce post method invocation dependencies. Table 3 shows different incoming messages to WSMS and actions it takes.

Table 6.3: Actions taken at WSMS

Tag	Action	Outgoing port
0 or 2- Method invocation from msgHandler	Check for workflow date (file access) and send the message to WSCMS to enforce pre execution dependencies	Send the message to wscms through “outtowscms” port.
5- From WSCMS after enforcing pre method execution dependencies.	Invoke the WS method	Send the message to ws through “outtows” port.
6- Results after method invocation from WS	Update state information and send the message to WSCMS to enforce post execution dependencies.	Send the message to wscms through “outtowscms” port.

6.3.3 The Web Service Coordination Management System (WSCMS)

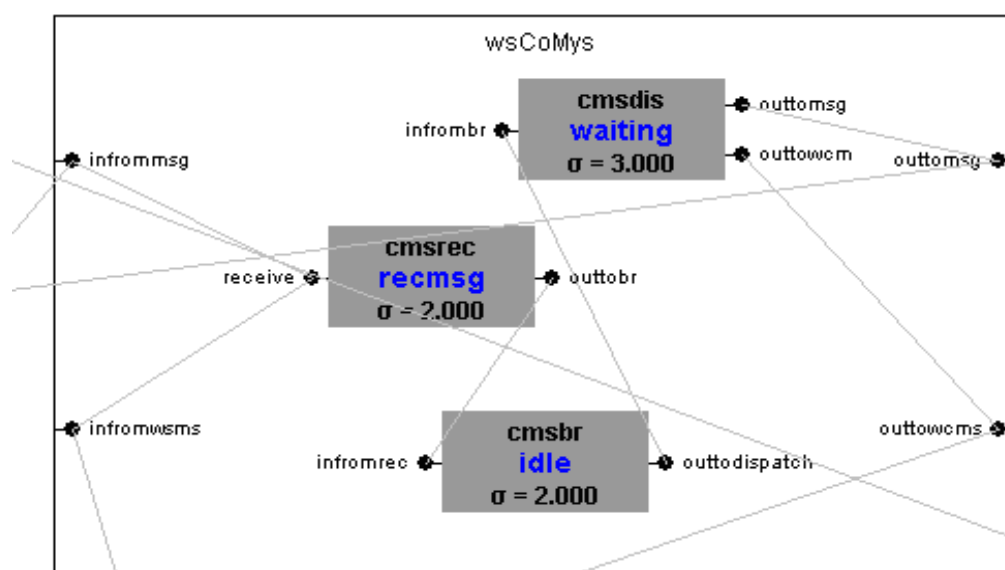


Figure 6.7: Web Service Coordination Management System

Similar to the message handler, WSCMS also consists of three components: a message receiver (**cmsrec**), a bond repository (**cmsbr**), and a message dispatcher (**cmsdis**) (Figure

6.6). Main component of the WSCMS is the bond repository. Each workflow, maintains its own dependencies in a file. Message receiver receives messages and puts them in a FIFO queue. Then it passes messages to the “Bond Repository” to take appropriate actions. In our system, we have employed web coordination bonds to model dependencies. Therefore, WSCMS essentially maintains web coordination bonds and manipulate them in order to enforce workflow dependencies. Dependencies are stored in the Bond Repository. Bond Repository is a persistent storage where each workflow has a corresponding bond store. Table 4 shows different messages it receives and corresponding actions of the bond repository. Upon completion of the action, it sends the message to the dispatcher and dispatcher directs the message to the appropriate component.

Table 6.4: Actions Taken at WSCMS

Tag	Action	Outgoing port
0 or 2- Method invocation from wsms	Check for workflow dependencies, change the tag to 5 (file access) and send the message back to WSMS	Send the message to wscms through “outtowscms” port.
1- From msgHandler to update dependencies (SB data)	Update the bond repository	
6- Results after method invocation from WSCMS	Check for workflow dependencies (post) (file access) and send the message to msgHandler	Send the message to msgHandler through “outmsg” port.

6.3.4 Web Service

A web service receives messages and invokes appropriate methods. After invoking the method, it changes the tag from 5 to 6 and sends the result back to WSMS. Method

invocation has been implemented as a “holdIn” time in the simulator. HoldIn method of the simulator allows us to wait for a particular time period at a defined state. For example, when a method is being executed, web service changes its state from “no-invocations” to “invoking” state. Table 6.5 shows different states of each component.

Table 6.5: Different states of Middleware Components

Module	Component	Initial State	State while processing
megHandler	mrec (message receiver)	active	active
	mres (message resolver)	waiting	resolving
	mdisp (message dispatcher)	waiting	dispatching
wsMgt	wcmsoh (object handler)	idle	active
	wcmssh (state handler)	idle	updating
wsCoMgtSys	cmsrec (message receiver)	recmsg	recmsg
	cmsbr (bond repository)	idle	updating/checking
	cmsdis (message dispatcher)	waiting	dispatching
Web Service	Web_Service	no_invocations	invoking

6.4 Simulation Scenarios

The first set of simulations has been carried out to verify following two scenarios.

1. Enforcing workflow dependencies using subscription bonds (post conditions)
3. Enforcing workflow dependencies using negotiation bonds (precondition)

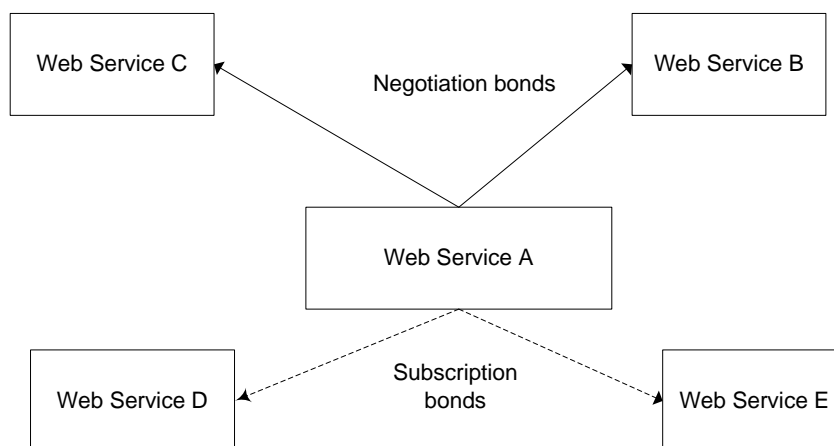


Figure 6.8: Simulation Architecture

Figure 6.8, further elaborates our simulation architecture. As we have explained in chapter 5, WSCMS ensures that pre-execution dependencies are met before making the web service method call. Series of events take place in local WSCMS as well as destination WS's coordination management systems while enforcing pre-execution dependencies. Figure 5.6 illustrates the interaction among WSCMM components while enforcing join-dependency constraints. Message handlers maintain an inbox and outbox for each workflow application. Both inbox and outbox has entries for each join-dependency point. When it receives control/data from destination entities message handler direct them for the appropriate inbox. Once the activity receives trigger (control) to perform the method call (step 0), it sends a message to the WSCMS for dependency check (step 1, 2, and 3). If all the dependencies are met web service method get invoked

and state information is updated (step 5). Otherwise, WSCMS sends messages to all the remaining destination entities for dependency check (step 4). Then, the remote web service invokes the corresponding method and sends the response to the requester web service.

Similarly, enforcing post execution dependencies require web service to trigger set of remote web services depending on the workflow constraints specified. Figure 5.7 illustrates the interaction among WSCMM components while enforcing post execution dependency constraints. As shown in Figure 5.7, WSCMS requests the message handler to send data/control to remote web services according the workflow split criteria. Message handler places remote invocations to the outbox (dispatcher) and triggers remote web service methods (step 6). At the same time coordination management system updates state information (step 5). We have simulated these two scenarios based on a method invocation in web service A as shown in figure

Simulation results in Table 6.2, show that middleware components behave in the correct order while running all of the above scenarios simultaneously. This indicates that our WSCMM middleware components successfully enforce negotiation and subscription bond dependencies.

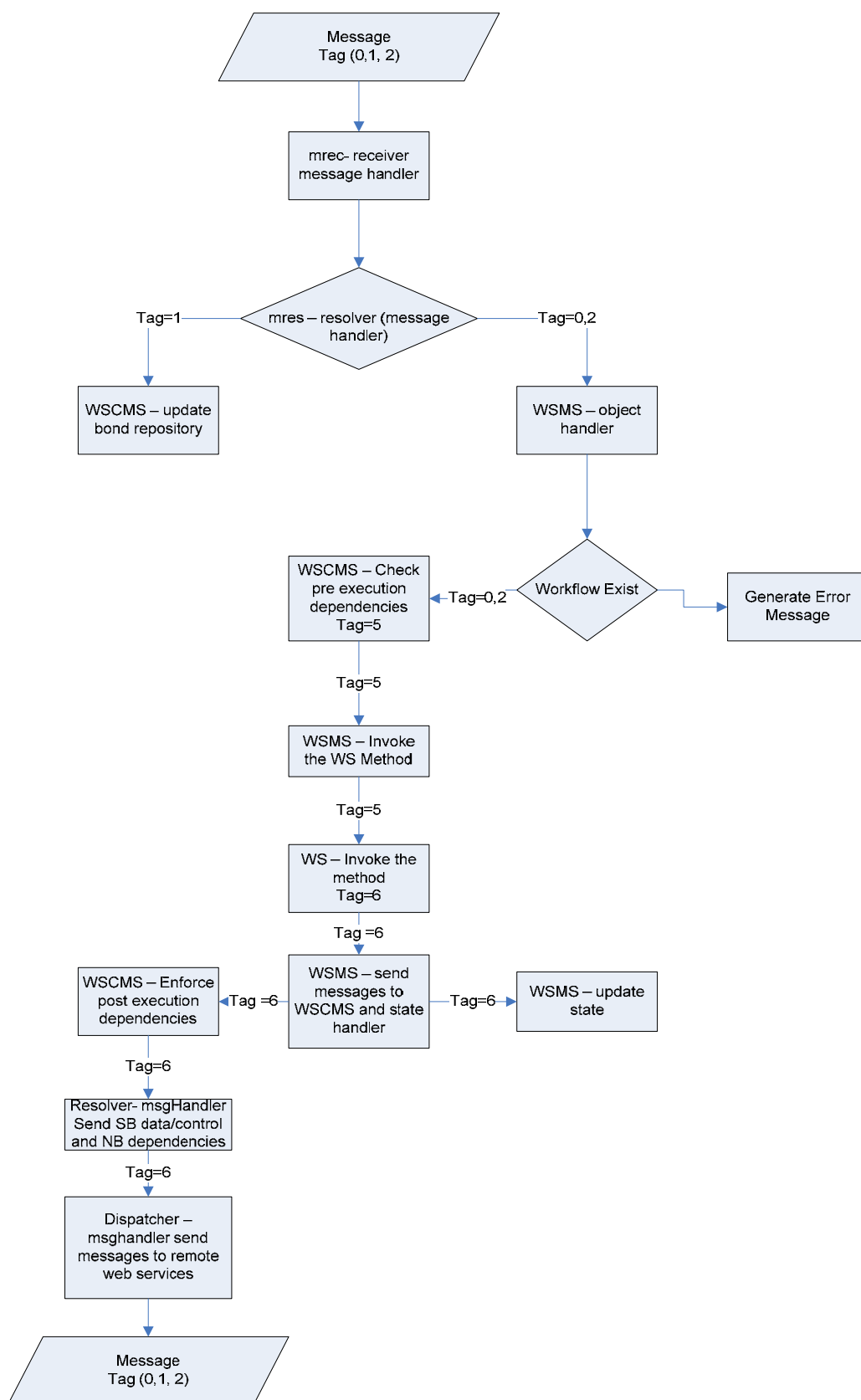


Figure 6.9: Message Routing in WSCMM Simulation

6.4.1 Simulating Pre-Execution Dependencies

Input: message : [w1:ws1:m1:p1:0](#) (Method invocation to A)

The first message belongs to workflow 1. It is from web service 1. Request web service A to execute method m1 with parameter set p1. Here, '0' indicated a method invocation. Figure 6.10 is a snapshot of pre-execution dependency simulation and the figure 6.11 is a snapshot at post-execution dependency simulation.

Output of the Simulation

Table 6.6: Simulation Output for Incoming Messages

Message Sequence	Description
sending message to the resolver msgHandler: w1:ws1:m1:p1:0 message received resolver: w1:ws1:m1:p1:0 resolving the message: w1:ws1:m1:p1:0	Case 1 at message handler to resolve the message.
message received resolver: w1:ws1:m1:p1:0 Method invocation, resolver: w1:ws1:m1:p1:0 Message Received at WSMS: w1:ws1:m1:p1:0	Case 1 at WSMS, identify as a method invocation
check dependencies before invocation, WSMS: w1:ws1:m1:p1:0 message received at Coordination Management System: w1:ws1:m1:p1:0 message received at Bond Repository: w1:ws1:m1:p1:0	Case 1 at WSCMS, identify as a method invocation and check pre execution dependencies (Figure 6.4)
Dispatch the message, resolver: wfsqu:App:m1:p1:2 (WS B) dispatch the message to remote WSs: wfsqu:App:m1:p1:2 (WS C) B and C receives messages	Send Messages to B and C to enforce pre-execution dependencies

message received msgHandler: wfsqu:App:m1:p1:2 message received msgHandler: wfsqu:App:m1:p1:2	
B and C send results to A dispatch the message to remote WSs: wfsqu:App:m1:p1:1 dispatch the message to remote WSs: wfsqu:App:m1:p1:1 A Receives results from B and C message received msgHandler: wfsqu:App:m1:p1:1 message received msgHandler: wfsqu:App:m1:p1:1	Receive response from B and C regarding pre-execution dependencies
Dependencies are met: case 0 w1:ws1:m1:p1:5 Bond repository updated:w1:ws1:m1:p1:0	WSCMS, pre execution dependencies are met and change the tag to 5
Message Received at WSMS: w1:ws1:m1:p1:5 Invoke WS Method, WSMS w1:ws1:m1:p1:5	WSMS, pre execution dependencies are met. Invoke the web service (Figure 6.5)
Message Received at WSMS: w1:ws1:m1:p1:6	WSMS after method invocation
Update state and post method invocation dependencies, WSMS w1:ws1:m1:p1:6 message received at Coordination Management System: w1:ws1:m1:p1:6 message received at Bond Repository: w1:ws1:m1:p1:6	WSCMS to enforce post method execution dependencies. Also, update the state information at WSMS
Send post method execution dependencies to D and E Dispatch the message, resolver: wfsqu:App:m1:p1:3 Terminated Normally before ITERATION 2 ,time: 128.0 Terminated Normally before ITERATION 2 ,time: 129.0 Terminated Normally before ITERATION 2 ,time: 130.0 dispatch the message to remote WSs: wfsqu:App:m1:p1:3 message received msgHandler: wfsqu:App:m1:p1:3 message received msgHandler: wfsqu:App:m1:p1:3	WSCMS, enforcing post method execution dependencies.
D and E receive message from A	D and E get subscription bond

message received msgHandler: wfsqu:App:m1:p1:3	based method invocations
message received msgHandler: wfsqu:App:m1:p1:3	

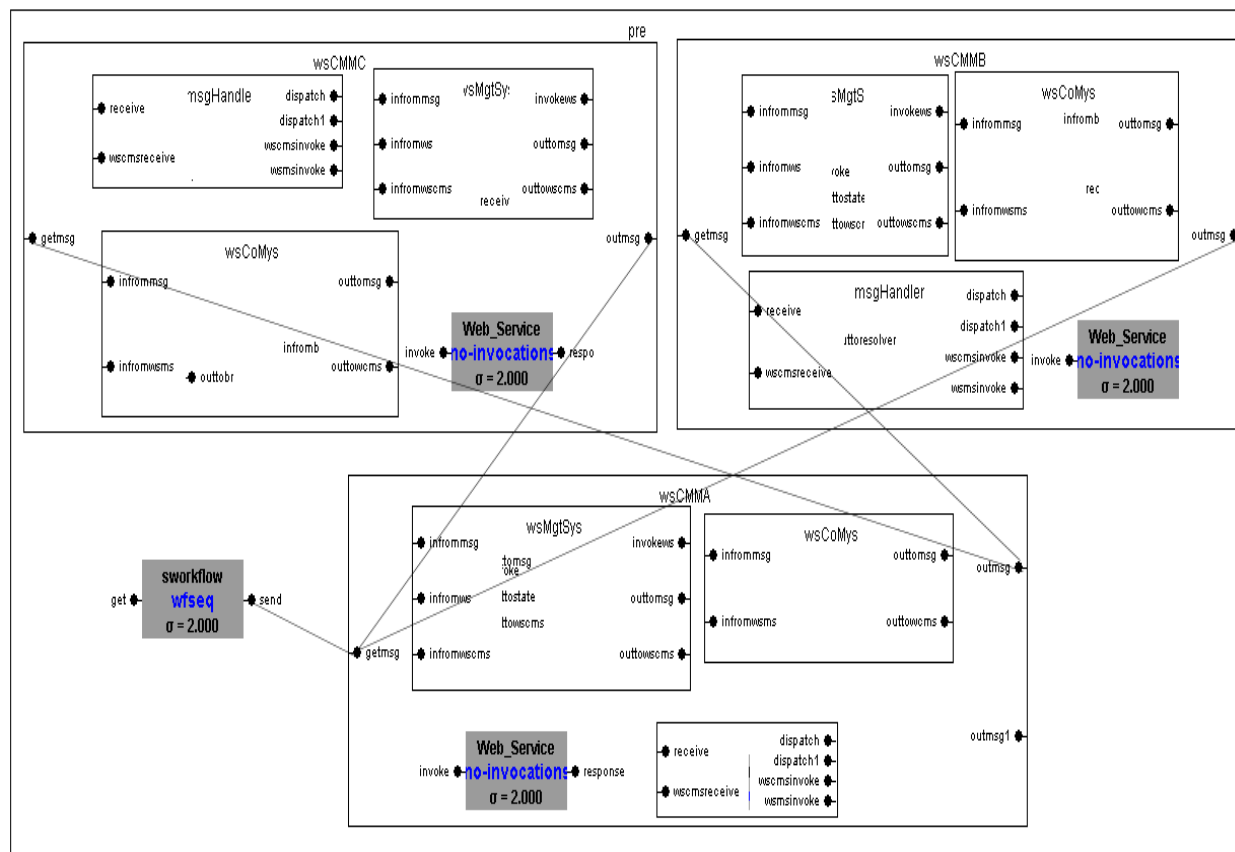


Figure 6.10: Enforce Pre – Execution Dependencies

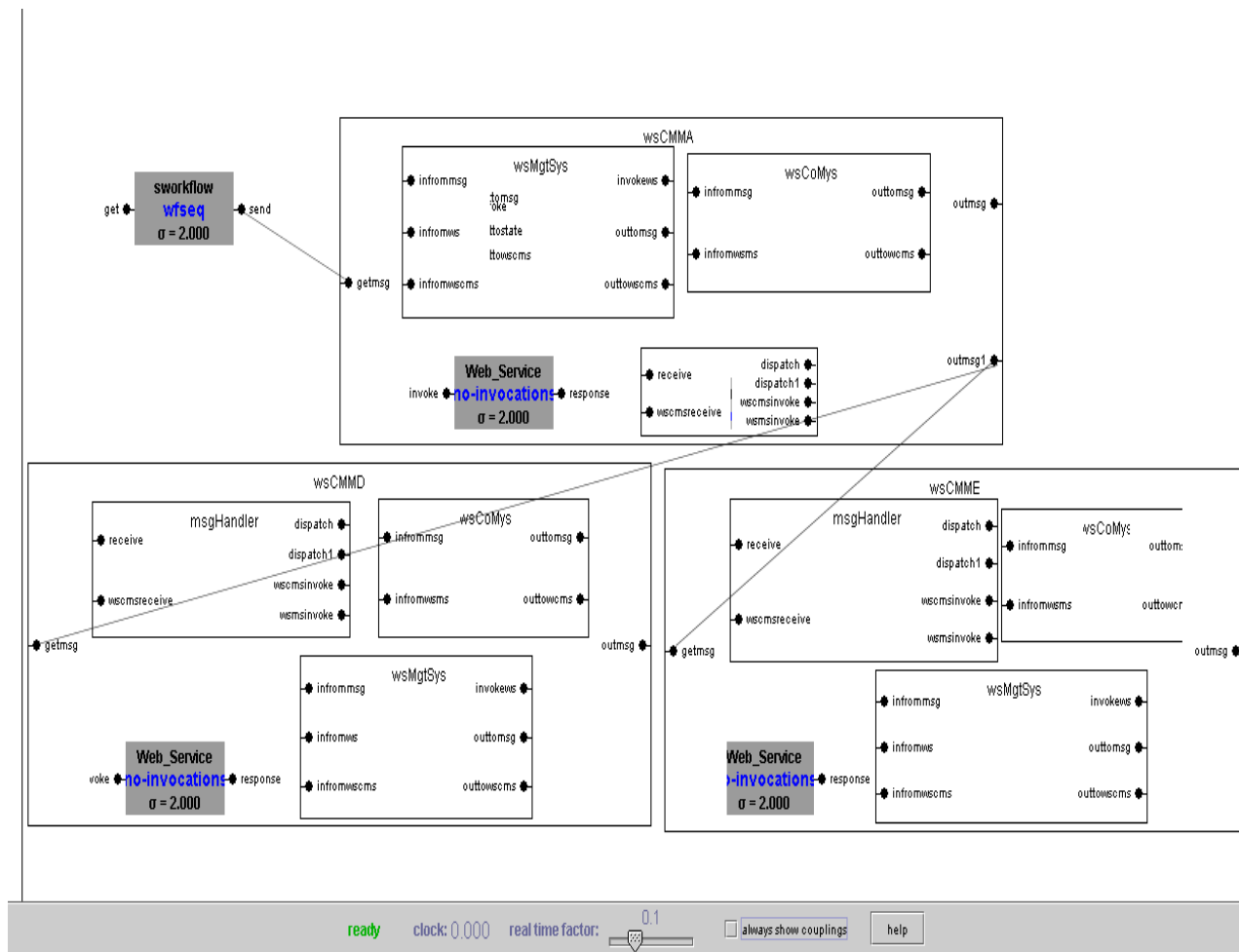


Figure 6.11: Enforce Post- Execution Dependencies

6.5 WSCMM: Compatibility with other Standards

In this chapter we proposed enchantments to the web services infrastructure which are analogous to the evolution of database application development and workflows. Our architecture consists of a management system and a coordination management system for web services. In fact, such an evolution is natural and verified due to the fact that current web service composition is a collection of several separate protocols to handle each functionality layer. For example, BPEL and WS-Coordination protocols handle application logic and coordination layers while WS-Transaction takes care of transaction management. Auxiliary protocols such as WS-Conversation and WS-Addressing have added capabilities to handle conversation (messaging) among participant entities and proper binding to web service ports (methods) effectively and efficiently. With these developments, currently there are two trends in web services composition.

- i) Develop the composite web process using a language such as BPEL. The use auxiliary protocols such as WS-Transaction, WS-Coordination, and WS-Addressing to add more functionality such as transaction management [Dus04, Tai04, Hul04]. This methodology results in heavily loaded composite web process having central coordination. Such code is difficult to manage and debug. Central coordination is also not desirable.
- ii) In contrast, one can develop the basic code required for the application using a language such as BPEL and use infrastructure support to handle coordination (Middleware) , conversation (Conversation controllers) and transaction (TP-Monitors, Middleware) [Alo04].

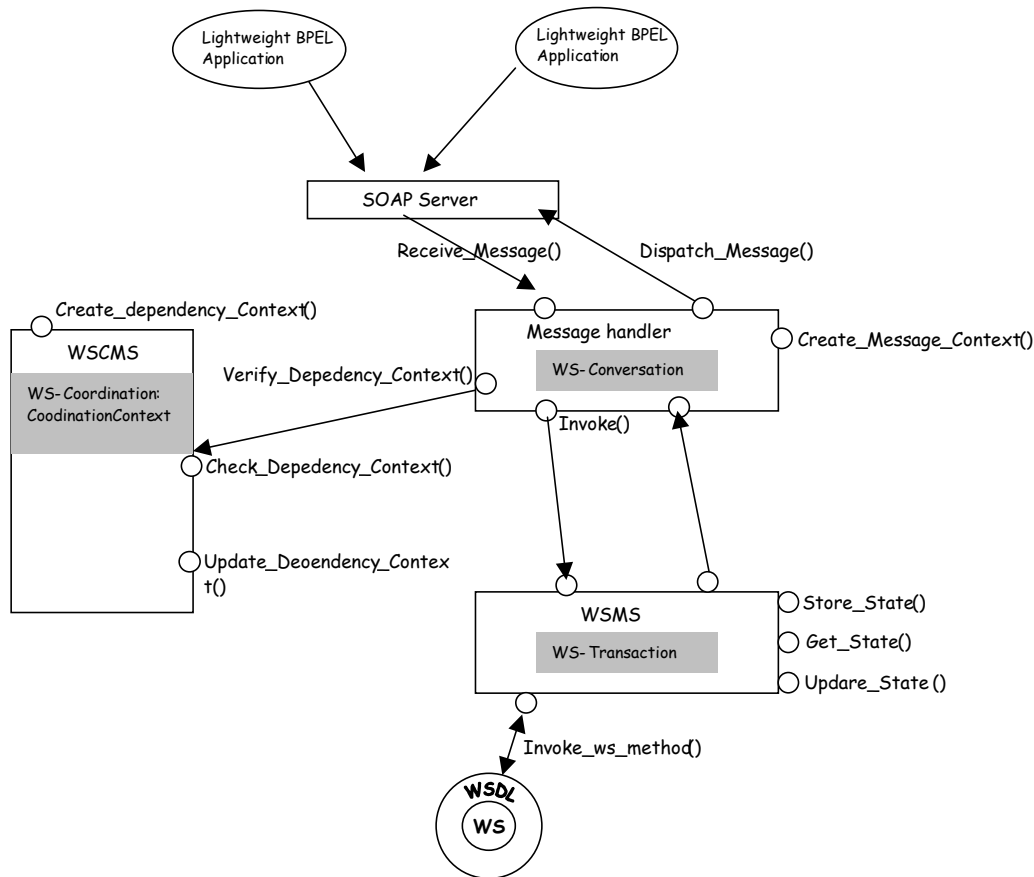


Figure 6.12: Web process architecture: Compatibility with other standards

We believe that the second methodology will have real impact on web services technology and help the evolution in more positive direction due to several reasons: i) distributed coordination, ii) scalability, and iii) lightweight application development. In our architecture we have taken this path. Earlier, we have described realization of our architecture using web coordination bonds. However, our architecture nicely fits into current web service composition and coordination protocols. As we have mentioned earlier, our middleware components have interfaces and interactions among components

happen through API's. However, the internal implementation, algorithms and data structures will be different based on the underline protocol being implemented. Figure 6.12 shows possible generic API's and interaction among components using current web service standards and protocols (BPEL, WS-Transaction, WS-Coordination).

As shown in Figure 6.12, the application logic can be coded using BPEL and all coordination, transaction, message handling, and state information are stored locally and managed locally. Here, BPEL code needs to trigger (start) transactions, but it does not need to handle coordination and workflow management functions. For example, "<scope> </scope>" construct will become simple and light weight. In our architecture, coordination context is an XML file containing all bond related information. Similarly, WS-Coordination creates the coordination context for each application using "Cretate_Coordination_Context()" method and manage it. Akin to the State Information, WS-Transaction creates TransactionContext for each transaction. WS-Conversation together with BPEL manages message correlation and sequencing similar to inbox and outbox in our conversation controller. Thus, the modules and our architecture are generic enough to accommodate current technologies. However, the internal implementation as well as the SOAP messages are being passed is different. Based on the protocol being implemented, it is necessary to have plug-ins and converters for inter operability. However, such plug-ins and converters will be light and simple because all these protocols use XML message data representation and SOAP messages for communication. Therefore, they are simple and much easier than RMI to CORBA or CORBA to DCOM conversion (Inter-operability).

6.6 Discussion and Related Work

Here, we critically discuss relevant important developments on web service composition, coordination, and enhancements to the basic web service infrastructure (Invoke/Response) in order to support proper coordination and composition without attempting to be exhaustive. Web services have become increasingly promising to solve barriers that the EAI (Enterprise Application Integration) communality faced for decades. In [9] authors have argued that Web services will play a major role in electronic data exchange and transaction processing systems. In [Ley02], authors illustrate how existing WSs are tailored to develop business processes over the Internet. Due to the service oriented nature of web services such applications need several web services to be integrated together to form a composed web process, in other words web service composition. Web services composition implies proper coordination (in particular control flow and dataflow) among participating web services to accomplish the business logic efficiently. Web service composition enables inter-organizational collaboration and coordination. Those coordinated activities are long running (workflows, transactions) and require much more functionality beyond just invoke-response protocols [Ley02]. In [Mue05], authors have pointed out the importance of integrating Web services in to workflow management systems. In [Men04], authors describe possible workflow application domains over the Internet. Application of workflow management systems (WFMS's) spans large number of application domains including business process models, scientific applications, and health care systems.

However, currently individual WSs are stateless and have no capability to store state information for long-lived transactions/workflows [Alo04, Bal05]. Participant Web

services are passive entities. Composition language and standards need to take care of application logic to transaction management. This resulted in heavy programming and will have negative effects towards the progression of web services technologies. Instead of having heavily loaded composition and coordination standards it is desirable to enhance the basic web service infrastructure (Invoke/Response) to support coordination and composition at web service level [Sch05, Jor05, Dou03]. Significant amount of research is being carried out towards this goal [Tai04, Ard03]. Table 3 presents a cross section of some of these technologies highlighting their goals. The last row illustrates our solution, web coordination bond-enabled web services which is discussed in section 5.

Table 6.7: Architectural Enhancements to Web Services

	Basic Service Description	Define Stateful Web Service	Transaction Aware	Communication Handling- Conversation controllers	Coordination Awareness (enforcing control flow/ data flow ..etc)	Session Management with service requesters	Peer to Peer communication (Distributed) *
WSDL [WSDL05]	Yes	No	No	No	No	No	No
WSCL [WSCL02]	Enhances WSDL	No	No	Yes	Not-Specified	Not-Specified	No
WSCI [WSCI02]	**	Partial**	Yes	Yes	Not-Specified	Yes	Yes
WS-Transaction [Lan03b, WST02]	**	Partial	Yes	Yes	No	Yes	Yes
WS-Coordination[Lan03b]	**	Partial	Yes	Yes	No	Yes	Yes
Self-Serv [Ben03]	**	Yes	Not-Specified	Yes	Partial	Yes	Yes
ServiceGlobe [Kei02]	**	Partial	Not-Specified	Yes	Not-Specified	Partial	Yes
WSTPM [Tai04]	**	Yes	Yes	Not-Specified	Yes	Not-Specified	Yes
Conversation Aware WS [Ard03]	Yes	Partial	Not-Specified	Yes	Partial- With the client not p2p	Yes	Not-Specified
Web Coordination Bond Enabled [Bal05a-b, Pra05]	Yes	Yes	Yes	Yes	Yes	Yes	Yes

* Web service to Web Service invocations, ** Use WSDL, *** Some state information only for supported features.

The web service description language (WSDL) [WSDL] describes the web service in terms of the operations it can support and of the protocols bound to such operations. However, even if the latest version of WSDL (2.0) specification has some improvements such as different interaction types defined, it lacks message sequencing and correlation capabilities. The Web Service Choreography Interface (WSCI) [WSCI] is an XML-based language, which starts from where WSDL (1.0) stops. WSCI describes the flow of messages exchanged by a Web service in the context of a process. WSCL [WSCL] provides a state-transition model for organizing the sequence of WSDL operations. However, it does not support context information, transactions, exception handling, message correlation, etc. However, WSCI provides a set of useful and necessary additions to WSCL. Another popular technology is WS-Coordination. The primary goal of WS-Coordination [Lan03] is to create a framework for supporting coordination protocol. This is achieved by standardizing a) A method for passing unique identifier between interacting Web Services (coordination context), b) A method for informing a protocol handler about port of web service that participates in conversation (registration), and c) A method for informing a protocol handler about the role it should assume in a conversation. WS-Coordination provides specifications for both centralized and distributed coordination. Conventional transactions and WS-based transactions are different in several perspectives. They have to work in a distributed setting resulting in often long-running. As lengthy business processes have to be executed, rigid ACID properties (atomicity and isolation constraints are relaxed) need to be relaxed. If the transaction is aborted, the web services execute a compensation operation rather than rollback. In order to tackle these issues WS-Transaction [Lan03] provides two types of protocols: a) Business activities for long-

running transactions, and b) Atomic transactions for short-duration transactions with strict ACID properties

The SELF-SERV [Ben03] project aims at providing tool support and middleware infrastructure for the definition and execution of composite Web services. They have prototyped a system in which Web services are declaratively composed, and the resulting composite services can be orchestrated either in a peer-to-peer or in a centralized way within a dynamic environment. The ServiceGlobe [Kie02] system provides a platform on which e-services (also called *services* or *Web services*) can be “implemented, stored, published, discovered, deployed, and dynamically invoked at arbitrary Internet servers participating in the ServiceGlobe federation” [Kie02]. One significant feature of ServiceGlobe is that constraints can be specified how many services should be invoked and how they should be invoked. Constraints may be specified directly when invoking Web services, but they may also be stored in a service's context. In [Ard03] authors propose to augment web services with message handling capabilities. They propose that each participant should store the conversation context and messages should be correlated and sequenced locally. Such conversation aware web services become active participants of the collaboration. Importance of adding autonomous behavior and self-manageability to web services has been highlighted in [Tai04]. Web service Transaction Middleware (WSTMW) [Tai04], is one such platform developed by IBM to carry out transaction over Web services. WSTMW resides in both Web service side as well as the client (mediator) side. They have employed WS-Transaction, WS-Policy and BPEL4WS to prototype the system. Also, the semantic web community has proposed an ontology-based framework OWL-S (DAML-S) to enhance the web service infrastructure [Ave02, Ver05, Bra03].

OWL-L proposes a new layer of metadata on top of WSDL so that it will add more semantics to web services. Such enhancements should strengthen the integration and composition and provide automatic verification mechanism [Hul04].

As we can see from Table 5, current systems are far from being complete. They propose many techniques (ad-hoc solutions). However, none of them are comprehensive enough to handle all the issues. Furthermore, a key challenge is to identify a minimal yet sufficient set of enhancements to web service architecture, both for reasons of efficiency and for better adaptability by the existing standards. All aforementioned systems propose different pieces of enhancements to the web services infrastructure. However, none of them are comprehensive enough to handle all the issues. Such proposals are in very early stage and warrant further extensive research.

6.7 Summary

In this chapter we argued that web services infrastructure need to be enhanced for effective distributed coordination over web objects including web services. Towards this goal we presented the web process architecture, a simple but powerful enhancement to the current web services infrastructure that transform passive stateless web services into conversation aware, stateful web objects. We strongly believe that such fundamental treatment is needed for further development of web services infrastructure towards achieving their original goal of seamless integration of autonomous web services for inter-organizational collaboration.

CHAPTER 7

THE BONDFLOW SYSTEM

Web Services have become the building blocks based on which new distributed applications will be created over Internet [9, Wee05]. Such applications span domains many domains including commercial application, scientific applications, and bio-medical applications. The enabling web services can be grouped into three broad categories [Tsu01]: a) simple web services (stock quote, traffic condition, weather), b) collaborative web services (decision making, hotel reservation), and c) transactional/B2B process integration web services (workflow, supply chain, process control). Typically, Simple web services are information providers. Interactions with simple web services are short-lived and synchronous communication protocols suffice. Collaborative and transactional web services provide building blocks to develop collaborative applications including workflows that may span inter-organizational boundaries. Such interactions are typically long-lived and require much more beyond just invoke/response protocols [Sch05]. Efficient technologies are required to rapidly develop and deploy robust collaborative applications leveraging off the existing web services. Three categories of users are envisioned who would be uses of the web services technology.

For example, travel reservation application and simple book purchase order workflow illustrate scenarios a common user will perform. Currently, these services are available as web portals. However, web portals are strict template level services where users are confined to predefined configurations. Ideally, more flexibility is desirable to select suitable services and configure them as per user's requirements. Using web service

technologies in scientific computing environments is increasingly becoming popular. Domains such as scientific biomedical applications (biomedical data, tool integration, and workflows) [Sin04, Var05], grid computing and even aerospace design and engineering use web service technologies [Alo04]. It is highly desirable for scientists to configure their workflows rapidly with minimum programming easily and effectively. Finally, expert commercial application developers (supply chain and manufacturing workflow) require modeling more complex control and dataflow dependencies that ensure transactional properties [Sch05]. Thus, such methodologies should empower common users, scientists and decision makers, and expert developers.

7.1 Limitations of Current Technology

Configuration: Common users and non-computer experts desire their workflows to be developed with minimum or no programming whilst having provisions for expert users to add more customizations. We denote the former as high-level configurability and latter as high-level programmability. Current technology lacks both of these features and they are either template level [Aal04] or detailed programming level [Wee05] systems. Template level tools lack flexible configurability while detailed level programming tools require the designer to model the workflow from scratch (ensure communication, workflow coordination, application logic, and transaction properties). Thus, intricate programming is required. In [Bar05], authors have pointed out the difficulty of using BPEL and WSDL especially for non-programmers and even with considerable efforts in web service standards still it is challenging to build non-trivial applications.

Deployment and execution platforms: World Wide Web became so popular due to its simplicity and easy accessibility. In contrast, CORBA, RMI and DECOM did not succeed as their proponents expected mainly due to the complexity of these technologies despite great features they carry [Wee05, Dus04]. Web services are to bridge the gap between two technologies. Therefore, ideally, applications that we configure using web services should be able to deploy and execute in web-like (preferably over Internet) infrastructure enabling them to be executed on both wired and wireless devices including servers, PCs, handhelds, and even on cell phones. Executing workflows over wireless devices has significant benefits [Dus04-Haw05]. Portions of long-running workflows can reside on handheld device providing monitoring and controlling capabilities as well as hosting services. Current web service workflow deployment platforms are difficult to interact with and confined only to expert users. Additionally, current platforms consume significant amount of resources and are difficult to deploy on limited resource wireless devices. Some of current web service composition and coordination architectures inherently assume that services are resident on the wired infrastructure. However, there is an increasing interest in both industry and academia to empower mobile devices. In [Cha04], authors describe issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition. A distributed architecture and associated protocols for service composition in mobile environments that take into consideration mobility, dynamic changing service topology and device resources are presented in [Cha04]. The composition protocols are based on distributed brokerage mechanisms and utilize a distributed service discovery process over ad-hoc network connectivity. In [Dus04] authors present architecture for mobile device

collaboration using web services. In [Mna04], authors present a rapid application to development environment for mobile web services. [Ste03, Haw05] present web service based mobile application integration framework. However, most of these technologies treat handheld devices as clients.

We designed the BondFlow system as a solution to the above problems. Underpinnings of the BondFlow systems are web coordination bonds and the WSCMM concepts.

7.2 The BondFlow Solution

Contributions of the BondFlow system are threefold.

1. Provide high-level configurability for non-experts while maintaining the high-level programmability for experts.
2. Distribute the coordination responsibilities among participating web services of the workflow by providing two distinct layers of functionality: Application logic layer and coordination layer.
3. Deploy and execute the workflow in platforms such as Internet using handheld devices so that the handheld device becomes the controlling/monitoring agent and possible service hosting entity.

Significance

Two layered workflow development methodology: Workflow coordination has been encapsulated in the BondFlow system as a separate functional layer using web coordination bonds. The web coordination bond is a fundamental underpinning of the

BondFlow system. Web bond coordination layer provide services to the application logic layer. This encapsulation enables the BondFlow system to hide coordination complexity from the developer. Developer's responsibility is to configure the workflow using high level constructs by linking web service appropriately and specifying constraints. Still expert developers can integrate programs to reflect complex interactions and constraints

Distributed coordination: We distribute the workflow coordination among participant web services by generating an “intelligent” web service coordinator proxy object (WSCP) or coordinator object for short per web service. These coordinator objects are stateful and enable encapsulated web services to be interconnected. An interconnected coordinator object together with its dependency parameters represents a coordination aware workflow node on behalf of the encapsulated web service.

Proof of concept working platform: The Bondflow system allows high-level configurability, high-level programmability, and distributed workflow coordination. The footprint of the BonFlow runtime is 24KB and the additional third party software packages, SOAP client and XML parser, account for 115KB. Moreover, the footprint of the coordinator object is small (~10KB) enabling them to reside on java-enable handheld devices. The intermediate system generated files are less than 100 KB for a sufficiently large workflow. The execution time workspace used by the BondFlow system is 5.4 MB including JVM (Jeode 1.2 handled java version). We have tested the BondFlow system on both wired and wireless infrastructure. We have used SOAP communication in wired devices and our SyD middleware in wireless devices. SyD is our recently prototyped

middleware platform to develop and execute application over handheld devices [Pra04a]. Lightweight SyDListener enable handled devices to communicate among application deployed on other peer devices.

7.3 Developer's View of BondFlow System

The BondFlow system initiates its operation by web service lookup and discovery (Figure 7.2). The web service (WS) interface module that contains WS locator helps discovering the service of interest.

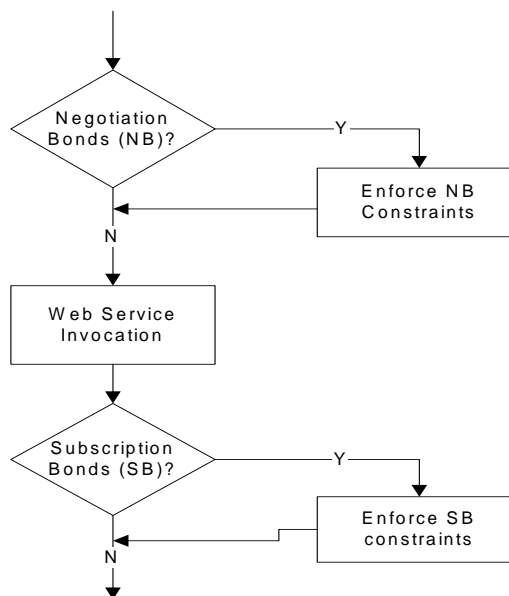


Figure 7.1 Flow within the Coordinator Object

The WSDL Parser parses the web service description and allows the service components to be viewed in the form of summary of methods and parameter list. Users can choose to save the viewed services for future reference. Instance of java-enabled web service coordinator object is created when the user wishes to save the web service. Web coordination bonds are created among the saved services to reflect workflow

dependencies. Dependency enforcement and entire operation of bond execution depends on the type of the bond that has been created. Bond related information is stored in an XML storage file. The CPO encompasses all the coordination capabilities of the web bond artifacts.

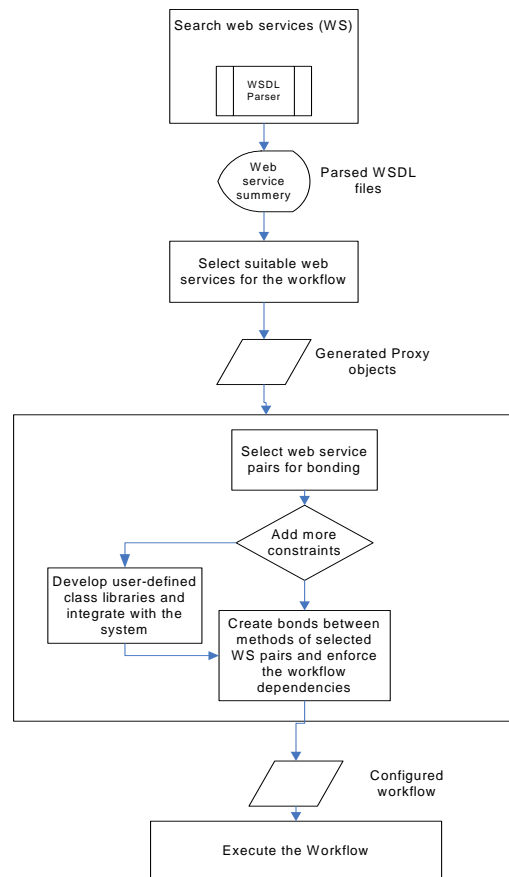


Figure 7.2: Developers Perspective of the BondFlow System

As shown in Figure 7.1, each web service call is encapsulated by a negotiation bond and subscription bond check. This logic makes sure that data and control dependencies are met before making the actual WS invocation. It hides the heterogeneity of various objects including legacy web services distributed among the network by enabling them to

coordinate using the BondFlow system. The bond coordination logic that the CPO contains is transparent to the user at all times. Once CPOs are created and bonded, the basic skeleton of web service composition for BondFlow system is ready.

7.4 Two-Layered Workflow Software Architecture

As shown in Figure 7.3a, the architecture of the traditional workflow code is “single layer” where developer needs to program the workflow from scratch (ensure communication, workflow coordination, and intermediate data processing) (Figure 7.3a). In contrast, in the BondFlow system, workflow coordination has been encapsulated as a separate layer using web coordination bonds. In addition, the system generates Java-based coordinator objects to represent participating web services in the workflow. The coordinator object encompasses all the coordination capabilities of web bond artifacts (Figure 7.3b). Coordinator proxy object communicates with the web service from method invocations and is state preserving. Capabilities of web coordination bonds including modeling workflow dependencies have been encapsulated in the upper layer (Figure 7.3b). Developer’s responsibility is to configure the workflow using high level constructs by linking web service appropriately and specifying constraints (high-level configurability).

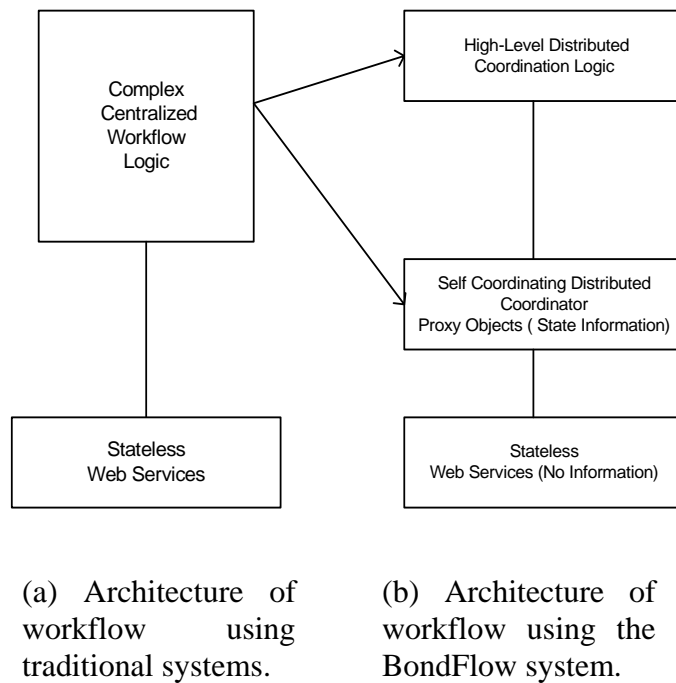


Figure 7.3: Two-Layer Workflow Software Architecture

Web Service Coordinator Proxy Object (CPO): Figure 7.4 illustrates components of the coordinator proxy object. The coordinator object provides the same interface as the web service provides to the outer world. Web service method invocations of the workflow take place through the coordinator object and the web bond coordination layer ensures that pre and post method invocation dependencies are satisfied. As shown in Figure 7.4, each coordinator object has a bond repository, a set of user-defined constraints (if any), and runtime information associated with it. The bond repository consists of all the workflow dependences related to the coordinator object (participating web service).

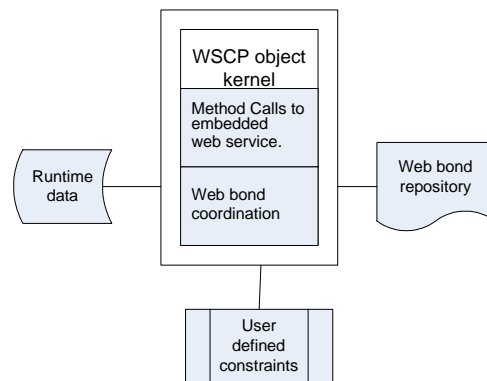


Figure 7.4: Web Service Coordinator Proxy Object

This indirection allows us to bring transparency to the system and hide the necessary coordination and communication logic behind it. It also maintains the status of method invocations such as intermediate data and partial results. User defined constraints represent the additional dependency conditions (dependencies not defined using web bonds) needed to be satisfied while enforcing workflow dependencies. User defined constraints have been discussed in section 7.2. As shown in Figure 7.5, each web service method call is encapsulated by a negotiation and a subscription bond check. The negotiation bonds enforce pre-method invocation dependencies while the subscription bonds enforce post method invocation dependencies.

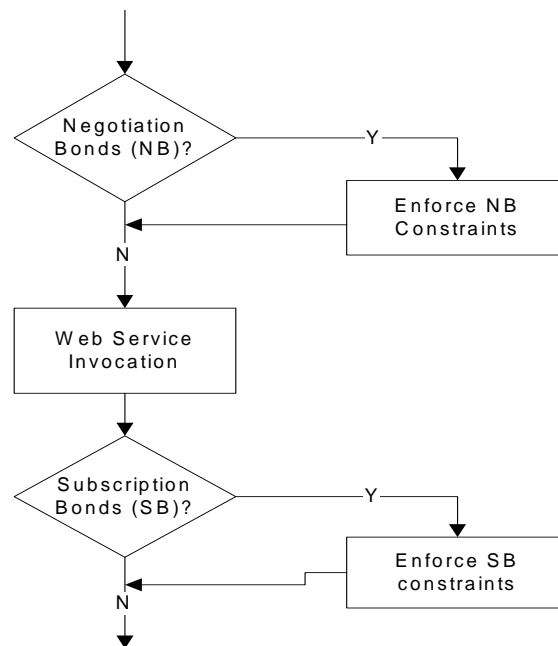


Figure 7.5: Flow within a Proxy Object

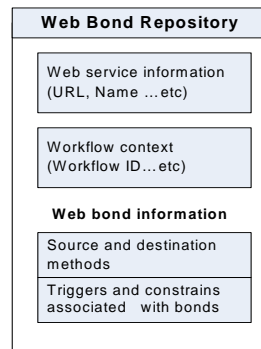
This logic ensures that workflow dependencies are satisfied with associated WS method invocation. For example, upon receiving an invocation, CPO requests the “Execution Module” to enforce pre-execution dependencies (enforce using a network of negotiation bonds). Consequently, the “Web Bond Manager” checks the corresponding bond repository and informs other coordinator objects to enforce the dependency (Figure 7.5). Here, enforcing dependency implies successful invocation of corresponding web service methods. Upon receiving the request, other objects check its runtime information (status of the method invocation - success or failure and intermediate data) and notify the status of the negotiation bond dependencies. The “Web Bond Manager” collects all the responses and informs the proxy about the outcome. Subsequently, the proxy object invokes the actual web service method; updates its runtime state information, and

enforces post-execution dependencies (enforce using a network of subscription bonds). In this architecture, each proxy object maintains and enforces workflow dependencies locally, allowing decentralized workflow coordination.

The idea of Web service coordinator proxy object together with underlying web bond primitives encapsulates the workflow coordination layer. This simple, but powerful idea empowers web services and makes workflow configuration less programming intensive. We believe this concept has enough potential to lead a fundamental shift in workflow development over web services.

7.4.1 Web Bond layer and the Bond Repository

The workflow configuration process starts by creating bonds among methods of selected web services to reflect dependencies (negotiation and subscription bonds). Bond constraints are specified during the bond creation time and the bond configuration is stored in a persistent storage in XML format.



```

</Wrapper>
  <WSName> <WSName>
  <!--Bond information is stored as follows
  between <Bond> </Bond> tag, source
  method name, destination web service and
  method name are the important parameter
  that are stored at bond creation along with
  type of bond and presence of trigger -->
  <Bond bid=" " >
    <SrcMethod> </SrcMethod>
    <DestWS> </DestWS>
    <DestMethod> </DestMethod>
    <Type> </Type>
    <Trigger> </Trigger>
  </Bond>
  
```

Figure 7.6: Elements of a Typical “Bond”

Figure 7.7: Sample Bond Repository

Repository

Figure 7.6 shows the structure of a typical bond repository. The bond data store (repository) consists of four elements. The first element is to identify the web service (hence the coordinator objects) the repository belongs to. The second element identifies the workflow/application to which the repository belongs. Source and destination methods and associated constraints among bonds are in the next two elements. A sample bond repository is shown in Figure 7.7.

7.4.2 Web Bond Layer

Here, we illustrate the workflow configuration using high-level web coordination bond constructs using purchase order case study workflow. Figure 7.8 illustrates the modeling of purchase order workflow using a network of web coordination bonds. Five web services are involved in the workflow. The system generates coordinator proxy objects for each web service. Then, a network of web bonds has been created among methods of these coordinator objects to enforce the workflow constraints. For example, the “receive purchase order” web service needs to pass control to “price calculation”, “find shipper”, and “production and shipment web services” once it is completed. In order to model this split-dependency, `Receive_PO()` method has three subscription bonds to each of `Initiate_PC()`, `Find_Shipper()`, and `Initiate_production()` methods. Similarly, rest of the dependencies has been modeled using other negotiation and subscription bonds.

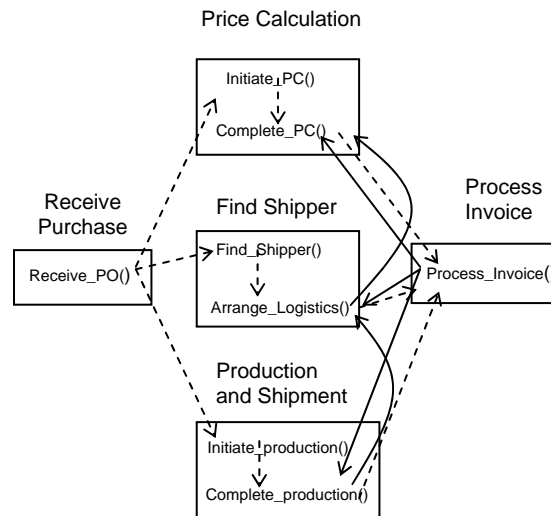


Figure 7.8: Purchase Order Workflow

The configured workflow consists of five coordinator objects representing each web service with bond repositories associated with them.

7.4.3 High-level Programmability

Simple workflow constraints such as AND-split can easily be enforced using web coordination bonds [Bar05]. However, complex control patterns such as “Sync-merge” and “Milestone” need developer designed selection criteria [Bar05]. Such customizations can be incorporated by developing user-defined libraries (java classes) and integrating them to the system library (typically complex workflow need such customizations). Then the triggers/constraints portion of the bond repository refers to the user-defined library (Figure 7.7). The BondFlow system is capable of extending the default web bond constraints allowing a plug-in architecture that extends the scalability of the system. Furthermore, it empowers the system’s ability not only to support the well known workflow patterns but also any arbitrary patterns to be created and deployed.

The extended bond constraints (user defined constraints) define one or more “Roles.” Each role performs a set of coordinating activities in order to enforce the semantics of the role. Furthermore, these roles are to be assigned to specific web services (nodes) in the workflow, thus allowing distributed coordination among this web services. The BondFlow system provides a common interface where new web bond constraints can be plugged-in. The extended bond constraints define a JAR file. This package contains: (i) *roles.xml*: This file contains definition of all the roles and their binding to specific constraints classes: (ii) *Set of class files*: These class files relate to each role defined in *roles.xml*. There are no restrictions as to the name of the class files. After preparing the JAR file, it is stored in the /plug-ins directory of the workflow configuration manager.

Once the workflow has been configured, it can be deployed on a single device or it can be distributed among several devices. They communicate with each other to enforce workflow dependencies. If the workflow resides in a single device, then the communication among coordinator objects is local in-memory calls. If the coordinator objects are distributed in the network, then SOAP or other suitable communication protocol can be employed to facilitate inter-object communication. We have implemented SOAP based communication in wired infrastructure and SyD middleware based communication in wireless infrastructure.

7.5 The BondFlow System Architecture: Design and Implementation

The BondFlow system consists of two sub-systems: workflow configuration manager and the workflow execution module. Workflow configuration manager consists of web

service interface module, WSCP generator module, and workflow configuration module. Workflow execution module consists of web bond runtime manager, SOAP or other suitable communication layer, and JVM runtime.

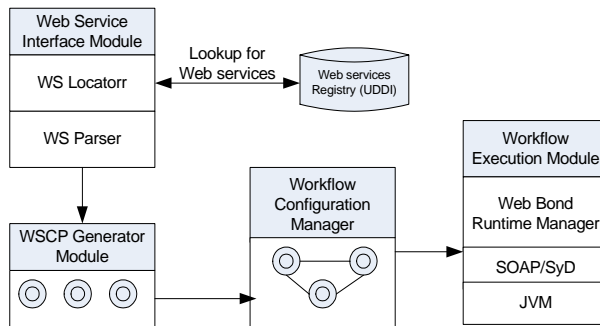


Figure 7.9: BondFlow System Architecture

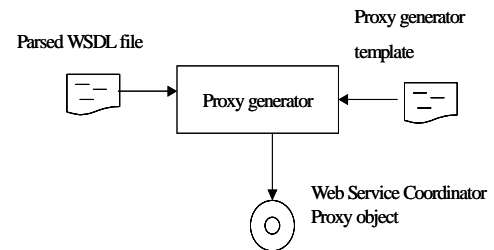


Figure 7.10: Proxy object generation

Configuration module:

Web Service Interface Module: The WS Interface module is the system's interface to the web services. It deals with locating the web services of interest for the user and parsing those web services for desired data. It consists of two components, Web Service Locator and WSDL Parser as shown in Figure 7.9. The web service locator module locates the service by contacting web service directory such as UDDI, gets the web service description and passes it to the WSDL Parser module. We have used Apache- Axis implementation of the web services. The WSDL parser uses WSDL4J API for WSDL parsing. It parses the WSDL file for required components and methods and parameter list is shown to the user for his reference. Parsed WSDL file is stored in the persistence

storage if the user opts to save the web service. Data is stored in XML format according to the bond repository schema.

Web Service Coordinator Proxy Generator Module: Upon selection of a particular WS for the workflow a coordinator object is generated. Coordinator object code is generated based on the parsed WSDL file of the selected WS and the proxy generator template (how do we generate what API's ...etc).

Workflow Configuration Module: the workflow configuration manager implements operations of the workflow configuration module. The responsibility of the configuration manager is twofold. First, it is responsible for all the bond related operations, such as creation, deletion and updating of the web bonds and generating the bond repository for each web service. Second, it allows expert users to add customized features to the workflow. This is one of the key modules in our system that guarantees high-level programmability for expert users. Collection of coordinator objects together with corresponding bond repository represents a configured workflow (Figure 7.11).

High-level programmability for expert users: The BondFlow system is capable of extending the default web bond constraints. Thus, allowing a plug-in architecture that extends the scalability of the system. Furthermore, it empowers the system's ability not only to support the well known workflow patterns but also any arbitrary patterns to be created and deployed.

The extended bond constraints (user defined constraints) define one or more "*Roles*." Each role performs a set of coordinating activities in order to enforce the semantics of the role. Furthermore, these roles are to be assigned to specific web services (nodes) in the workflow thus allowing distributed coordination among this web services. The BondFlow

system provides a common interface where new web bond constraints can be plugged-in. Moreover it also provides the developer with a set of APIs, which can be used to gain access to the runtime of the system. These features of the system greatly reduce the development time. This set of APIs and interface are defined by classes and interfaces defined in Pattern package in the class hierarchy.

In terms of implementation, the extended bond constraints define a JAR file. This package contains:

roles.xml: This file contains definition of all the roles and their binding to specific constraints classes.

Set of class files: These class files relate to each role defined in roles.xml. There are no restrictions as to the name of the class files.

After preparing the JAR file, it is stored in the /plug-ins directory of the workflow configuration manager.

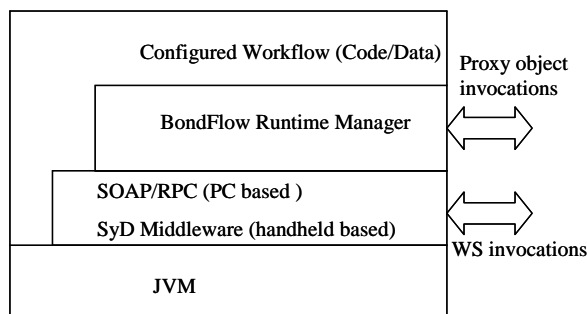


Figure 7.11: The BondFlow Runtime

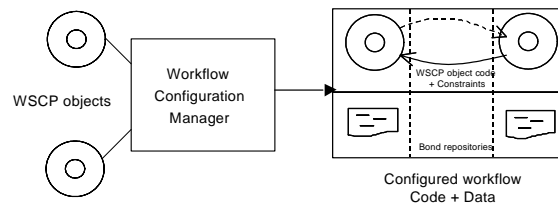


Figure 7.12: Workflow configuration

BondFlow Runtime: The BondFlow runtime consists of two modules: web bond runtime manager and the runtime information handler. The BondFlow runtime manager enforces workflow constraints at runtime whilst runtime information handler stores

method invocation information and other workflow related dynamic information for long-lived workflows. The BondFlow runtime manager sits on JVM and uses SOAP or other suitable communication technology to communicate among coordinator objects and web services. Upon object invocation, it consults the workflow execution environment and carries out series of operations depending upon the bond parameters specified at the bond creation time. Checking of the type of bonds, getting bond parameters and executing the actual bond are some of the major operations by the bond flow runtime manager. The final call to the original web service is made using SOAP or any other suitable communication standard. For example, if the coordinator object and the web service reside in the same location web service calls are in-memory invocations. Upon receiving an invocation, WSCP object request the “Web Bond Runtime Manager” to enforce pre-execution dependencies (enforce using a network of negotiation bonds). Consequently, the “Web Bond Manager” checks the corresponding bond repository and informs all remote proxy objects to enforce the dependency. Upon receiving the request, remote objects check its runtime information and notify the status of the negotiation bond dependencies. The “Web Bond Manager” collects all the responses and informs the proxy about the outcome. Subsequently, the proxy object invokes the actual web service method, updates its runtime state information, and enforces post-execution dependencies. Likewise, the coordination continues.

7.6 Handheld-Based Execution

The workflow applications have been executed on HP's iPAQ models 3600 and 3700 with 32 and Pra05 MB storage running Windows CE. There are two possible deployment

strategies. First, the entire workflow can reside in a single wireless device. In this case, communication among coordinator objects is via local in-memory calls. Actual web service call is made using SOAP (kSOAP). Second, the workflow can be distributed among several iPAQ's (Figure 7.13). This scenario is important in cases where some portions of the workflow can be monitored and executed by a selected set of users on specific devices and/or with specific security settings.

In this case, coordinator objects need to communicate using a remote messaging system to enforce dependences. We have employed the SyDListener of the SyD middleware [Pra04a]. The SyDListener enables handheld devices to communicate among applications deployed on other peer devices (Figure 7.10). SyDListener is a lightweight module in our SyD middleware framework for enabling mobile devices to host server objects. In order to communicate using SyD listener, first coordinator objects need to be registered in the SyD directory. SyDDirectory maintains its own database to store information about all the SyD application objects together with associated devices and delivers location information of devices and services (methods) dynamically. SyD objects can lookup remote objects through SyDDirectory. The SyDEngine facilitates the object to actually invoke a remote object. SyDListener keeps listening for any connection requests and delegates the control to the SyDEngine module.

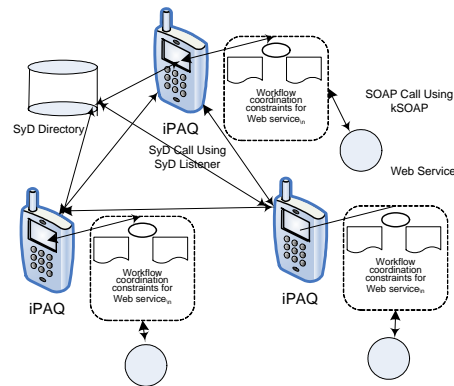


Figure 7.13: Workflow Distributed among Several iPAQ's

Coordinator Object Registration as a SyD Application Object [Pra04a]: The proxy objects register all the method names along with the list of parameters (their data types) with the registry. Initially, all the entities are converted into required XML format using SyDDoc and then the registration process with SyDDirectory begins. Once bound in the registry, these coordinator objects wait for invocation from other proxies. In this scenario, the registered proxies act as servers waiting for invocation from clients.

Coordinator Object Invocation through SyD Engine [Pra04a]: When a workflow containing SyD coordinator application object encounters the presence of web bonds with other applications, it looks up the desired web service proxy in the SyDDirectory (Figure 7.9). SyDDirectory returns the list of parameters for the specified method. Depending upon the parameters, required values are passed to the SyDEngine as an XML document. The SyDEngine of the client (in this case the source web service) invokes its SyDListener that in turn calls the server's SyDListener by opening a socket connection. The result is returned to the client as an XML document. In this architecture, each device can act as

both server and the client. They become capable of hosting server objects. As shown in Figure 7.13, Actual web service call is made using SOAP (kSOAP).

7.7 System Evaluation

The BondFlow system has been prototyped using java 1.4 and the footprint of the BonFlow runtime is 24KB. Additional third party software packages, SOAP client and XML parser, account for 115KB. Non-device resident configuration module is 28.7 KB. The footprint of the proxy object is small (~10KB) and typically increases by 0.3 KB per additional operation (method) of the web service. Intermediate system generated files are less than 100 KB for a sufficiently large workflow. Typically the footprint of the bond repository increases 0.3 KB per each additional bond. The execution time workspace used by the BondFlow system is 5.4 MB including JVM (jeode handled version).

We have developed several workflows to evaluate the BondFlow system. We have used real web services available in xmethods.com and few other service directories for these workflows. Reminder of this section presents our system performance details.

Hardware software setup: We ran our experiments on a high performance SunOS 5.8 server. We built wrappers using JDK 1.4.2. The WSDL parser has been built using WSDL4J API. WSLD4J API is an IBM reference implementation of the JSR-110 specification (JavaAPI's for WSDL). NanoXML 2.2.1 is used as the XMLparser for JAVA. Various publicly available web services including Xmethod's SOAP based web services (<http://www.xmethods.net/>) have been used for our experiments. For wireless device experiments we have used HP's iPAQ models 3600 and 3700 with 32 and 64 MB

storage running Windows CE/Pocket PC OS interconnected through IEEE 802.11 adapter cards and a 11 MB/s Wireless LAN. Jeode EVM personal Java 1.2 compatible has been employed as the Java Virtual Machine.

Size of WSDL (number of methods) vs. Wrapper creation time: As Web bond wrapper is central to our system it is important to analyze wrapper creation time and to investigate how wrapper creation time varies with different size (number of methods) of Web services. Table 7.1 shows that wrapper creation time is very small and wrapper size is less than 10 KB even for a Web service with 17 methods. This is an advantage as these wrappers can easily be placed in memory constrained small handheld devices. The bond creation time for both types of bonds is less than 25ms. Also, note that once wrappers are created and bonded, the basic skeleton of the workflow is ready. Developers can add more logic into it if needed. This will reduce the programming effort considerably.

Table 7.1: Size of WSDL (number of methods) vs. Proxy Object Generation Time

Number of Methods	WSDL Size (KB)	Proxy Creation Time(ms)	Proxy Object Footprint (KB)
1	2	20	2.2
2	2.3	23	2.3
4	8	32	2.6
5	12	40	2.5
8	16	46	3.5
17	32	76	5.7

Case study workflows: We have developed few simple and complex workflows to evaluate the BondFlow system. Figure 7.14 and 7.15 show *book price* and *traffic condition* workflows respectively. Both of these workflows enforce simple sequence. As shown in Figure 7.14, there are subscription bonds from Barnes and Nobel web service to

eBay web service and eBay to Amazon and Amazon to Currency web service. This chain of subscription bonds enables them to exchange book price data and control. By having negotiation bonds in reverse direction make sure they activate sequentially. For an instance, Amazon can only be invoked if eBay has finished its activity. Similarly Figure 7.15 illustrates the bond structure for traffic condition workflow.

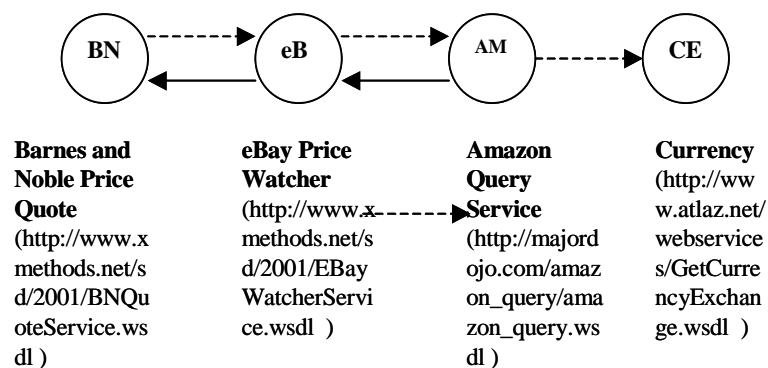


Figure 7.14: Book Price Workflow

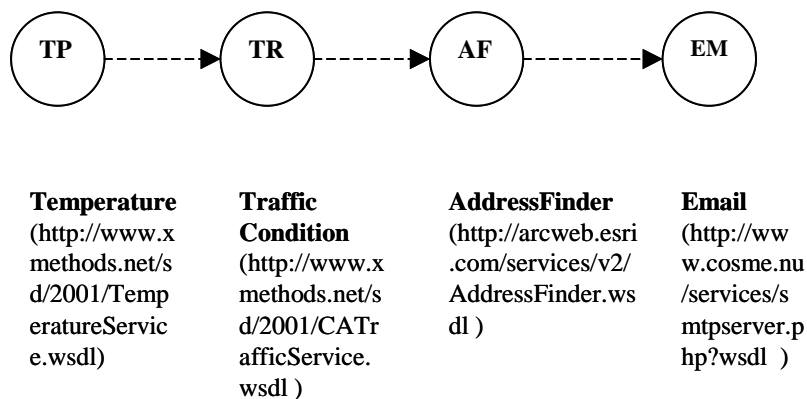


Figure 7.15: Traffic Condition Workflow

Then have developed several workflows to evaluate the BondFlow system. Here, we illustrates the online book purchase workflow and purchase order workflow.

Online book purchase workflow: As shown in Figure 7.16, “Start_Book_Purchase()” method sends control to both BN and eBay web services to get book quote (parallel split). Result is fed to the currency exchange web service where each quote is converted to the local currency. Then if the user is online send an email. Note that the currency exchange activity is invoked only if both BN and eBay book quotes have been completed and the user is online. This is captured by three negotiation bonds from currency exchange activity to each activity with AND logic.

Purchase order workflow: On receiving the purchase order the receive purchase order initiates three concurrent tasks to initiate the price calculation, select a suitable shipper, and scheduling the production and shipments. Once all three tasks are done, invoice processing starts task is initiated. We have modeled and implemented this workflow using the BondFlow framework. Figure 7.16 illustrates the modeling of purchase order workflow using web coordination bonds. Similarly, we have modeled several other workflows and carried out various performance measurements. Rest of this section discusses results of performance measurement.

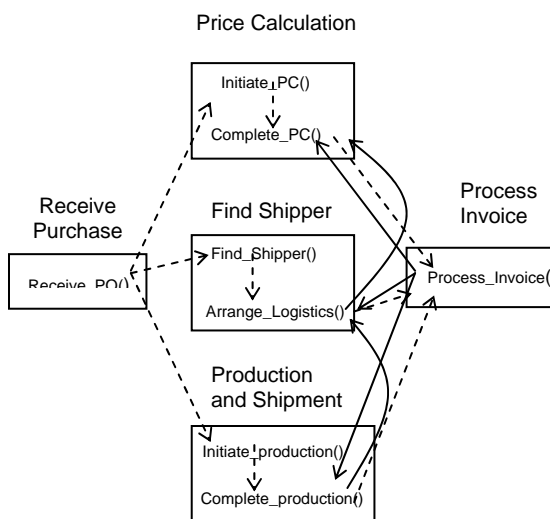


Figure 7.16: Purchase order workflow

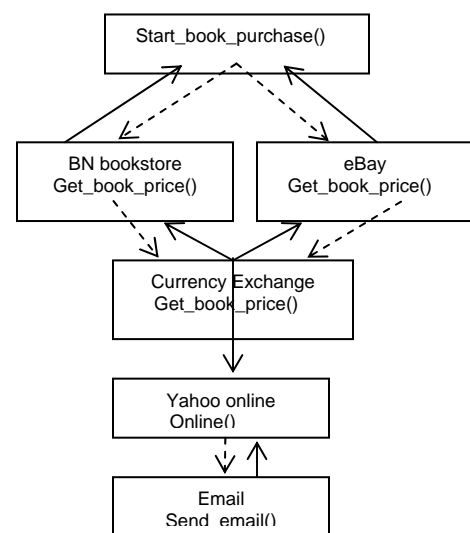


Figure 7.17: Online book purchase workflow

Table 7.2: Workflow execution timings

Workflow	Total execution time (ms)	BondFlow related time (ms)	BondFlow related (%) computation
Purchase order # of NB= 4, #of SB= 9	7820	1048	13.4
Online book purchase # of NB= 5 #of SB= 6	2483	102	4.1
Book Price #of SB's = 3, #of NB's= 2 (book price)	5577	82	1.4
Traffic Condition #of SB's=4	6406	67	1.07

Table 7.3: Footprint of the workflow

Workflow	Bond repository (KB)	Proxy objects (KB)	Total workflow (KB)
Purchase order	7.10	25.4	32.5
Online book purchase	5.82	19.8	25.62

We have deployed and executed case study workflows including the purchase order workflow on both wired and wireless infrastructure. Table 7.2, shows that the workflow execution timings for the two case study workflows for both wired and wireless settings. Bond related time for both workflows are approximately ~10% of the time without the BondFlow system. The bond related time accounts for times taken to check workflow dependencies in bond repository and initiate appropriate method calls on remote web services (coordinator objects). Table 7.3 shows the footprints of two workflows. The coordinator objects and corresponding bond repositories accounts for ~25% and ~75%

respectively. The footprint of the proxy object is small (~10KB) and typically increases by 0.3 KB per additional operation (method) of the web service. Intermediate system generated files are less than 100 KB for a sufficiently large workflow. Typically the footprint of the bond repository increases 0.3 KB per each additional bond. Thus, we feel that with in a very small amount of additional storage for the proxy objects, we have been able to get substantial gains in the speed of the workflow.

Benchmark Workflow Patterns: Finally, Figure 7.18 shows the execution timings for few different workflow benchmark patterns. Time taken in wireless setting is more mainly due to limited processing power and other resources. Also, the execution time rapidly increases with number of nodes. This is again due to the XML parsing.

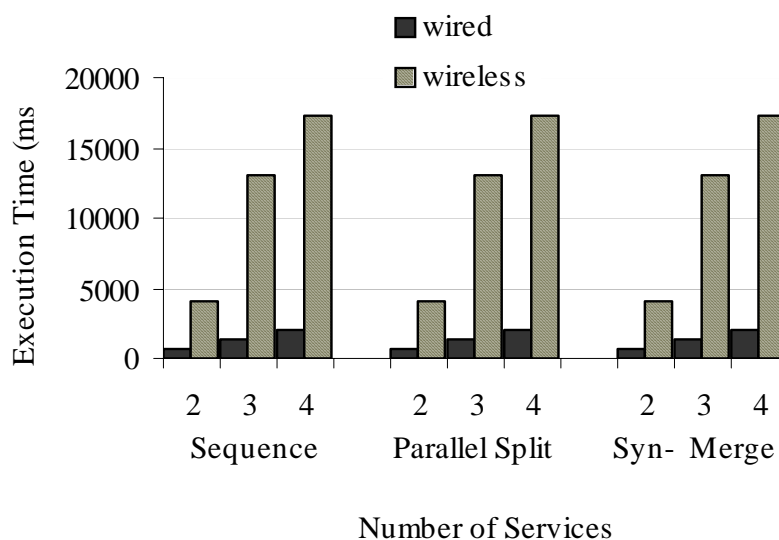


Figure 7.18: Execution timings for sample workflow control flow patterns

7.8 Related Work and Discussion

Several approaches have been proposed toward distributed web service coordination and peer-to-peer interaction among web services. Among such systems, IBM symphony [Gir04] decentralizes the coordination by partitioning centralized workflow specification into separate modules so that they can run in a distributed setting. However, there are limitations to such efforts. First, it is necessary to develop the centralized BPEL code and then partition and distribute it among participant entities. Second, usually, there are problems partitioning the code in complex application scenarios such as long running transactional applications without proper infrastructure support. Self-Serv project presented in [Lan03], proposes a peer-to-peer orchestration model for web services. It introduces a "coordinator," which can act as a scheduler for participating web services. Several coordinators can control the execution of the workflow in peer-to-peer fashion. In [Chr04] authors propose a distributed and decentralized process approach called OSIRIS that allows peer-to-peer communication among participating web services. However, their approach needs meta information to be stored in a central location. Also, in order to enforce fork/join dependencies they introduce a new join node exclusive from workflow nodes. In contrast to the Self-Serv and OSIRIS approaches, our coordinator proxy object is dynamically generated based on the description of participating web service and it encapsulates all the coordination capabilities. The proxy object enforces its own dependencies. This enhances each web service facilitating more fine-grained decentralization of the coordination. In [Sch02], authors propose a system to distribute the execution of business applications using web services by adding business rules into the SOAP messages. Business rules encoded in the SOAP header specify the order of

execution. Messages are decoded and processed by special processing units called SOAP intermediaries. In [Ros05], authors propose a service-oriented distributed business rules system and its implementation based on WS-Coordination. Web service Resource framework is another proposal towards stateful web services. It provides standardization representation to stateful resources and the web service interface provides functionalities to access (read, update and query) state information. This state information is used to process web service messages [Hum05]. Comparative study of various implementations of WSRF is presented in [Cza04]. In contrast to WSRF approach, in the BondFlow system maintains state information of workflow execution and processes messages. State is attached to the coordinator proxy object. Web service interface need not be changed and web service is relieved from state handling functionalities.

In [Cha04, Jor05], authors describe issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition. The composition protocols are based on distributed brokerage mechanisms and utilize a distributed service discovery process over ad-hoc network connectivity. In [Ran04], authors present an architecture for mobile device collaboration using web services. In [Mna04], authors present a rapid application development environment for mobile web services. [Ste03, Haw05] present web service based mobile application integration framework. However, a key limitation of most of these technologies is that they treat handheld devices only as clients.

7.9 Summary

In this Chapter, we have presented the design and a prototype implementation of our BondFlow system, which is a platform to configure and execute distributed workflows over web services. BondFlow system's two-layered workflow software development methodology greatly reduces the application development effort. The concept of the coordinator proxy object is central to our decentralized architecture. A preliminary study of implementation prototype shows that the bond related time is ~10% of the workflow execution time. Also, the small footprint of coordinator proxy object (~10KB) enables them to reside on java-enabled handheld devices. In contrast to other systems such as Self-Serv, the idea of the coordinator proxy object enhances each web service facilitating more fine-grained decentralization of the coordination. Our goal is to use this infrastructure to model and implement actual workflows in typical biological and E-commerce applications.

CHAPTER 8

BIOLOGICAL WORKFLOWS

Integration of data sources/tools and perform computations on them is one of the key areas of experimental biology. Modern data sources and computational tools are diverse in nature and many such sources are available. For example, according to [Hul06], there are about 3000 publicly available services in molecular biology itself. Moreover, these sources are geographically distributed and highly diverse in data format, representation, and capabilities. Therefore, manual composition and analysis has become almost impossible [Gua03]. Efficient and robust tools/methodologies are needed to automate the biological data and tool integration. Recently, web service technology has gained considerable recognition in both industry and academia as a possible solution to many such problems. In this chapter we illustrate how the BondFlow system can be used to compose biological data sources and tools to create useful workflows. First, we discuss challenges in biological data and tool integration in detail. Then, we present a detailed discussion of web services based tools for biological data and tool integration. Next, the BondFlow based solutions are presented. Finally, we discuss strengths and weaknesses of the BondFlow system and future directions.

8.1 Challenges in Biological and Data and Tool Integration

Modern biological data analysis requires the aggregation of many tools and data sources developed by various independent organizations [Atl04]. Such analysis involves data exchanges among different tools and execution of these tools in a particular order. This

essentially creates a workflow among participating entities. For example, DNA sequence analysis is one of the most popular workflows often biologists compose. In this workflow, first, a BLAST query can be made to extract matching sequences and then a query can be made to GetEntry data base to extract sequences of all the matching DNAs. Finally, a query can be made to ClustalW for multiple alignments. Such a data and tool integration differs from conventional commercial applications in several ways. Significantly large number of tools and data sources data sources available representing highly diverse and heterogeneous sources. Also, these data sources and tools are autonomous and have different interfaces and querying capabilities. For example, Genome research projects generate enormous quantities of data from a large number of high quality sequence data of different species and variants due to the advent of new and improved sequencing technologies [Att99]. There exist many standalone databases including EMBL at the European Bioinformatics Institute (EBI), the DNA Data Bank of Japan (DDBJ) at the Center for Information Biology (CIB) and GenBank at the National Center for Biotechnology Information (NCBI), which harbor such sequenced data and are goldmines for a biologist, especially for homology sequence comparisons and sequence analysis [Ben03, Ben00]. Moreover, data being transferred from one tool to another can be large and complex. Intermediate data conversion mechanisms are needed. Biological workflows can be long running and require more resources than conventional commercial applications. However, they may not require more complex workflow control flow requirements. In general, scientific workflows are data flow driven.

Currently, most common type of data and tool integration methodology for biologist is web based tools. However, web portals require significant amount of manual interactions

such as manual copy and paste data from one source to another. Also, in cases where large amount of data to be retrieved and analyzed this method is very inefficient if not impossible. Another technique is to use scripting languages such as Perl to compose these tools and data sources. They require expert knowledge of systems and skills. As many researches have pointed out, web services provide solutions to some of these problems. In the next section we explore web service based solutions to biological data and tool integration.

8.1.1 Web service Enabled Biological Tools

As mentioned earlier, such enormous data crunching requires the integration and mining of ever increasing heterogeneous bio-logical data sources into a desired configuration, which is effectively setting up a workflow among these data sources. Such integration and configuration needs to overcome the same issues that enterprise application integration technologies are faced with for decades on a different scale with added constraints such as data conversion and extracting most accurate data among different data sources. Moreover, any such integration and data mining tool should be user friendly and transparent to the user as much as possible. Web services have emerged as a capable platform, which hides system and network heterogeneity issues making users as well as the application developers life easier. In [Sha04] authors have mentioned several advantages of using web services in biological data crunching.

1. They are universally interoperable because of language independent protocols such as WSDL, SOAP, and XML.

2. They have a simple way of communication using loosely coupled SOAP messages.
3. Developers, users do not need to perform any code or any installation procedures. Web services are distributed across the network and users can build their applications using well understood protocols such as HTML, and XML.

Currently, many web service based systems available for scientific workflow composition and execution. One of the prominent objectives of all of these systems is to facilitate non-computer experts with “some kind of” easy use graphical workflow configuration environment. Among such systems, BioFlow has a well designed architecture [Gua03]. The main objective of the BioFlow system is to facilitate seamless integration of online distributed data sources and programs. BioFlow supports query based workflow composition. It consists of five sub components. Its program integration module facilitates inter-program communication. For example, if programs are running in the same computer then the interaction is takes place through OS calls. Otherwise, suitable Remote Procedure Call mechanisms need to be used. Its data integration module supports inter-data source (DB) communication through a query language called HTQL. Inter program data conversion is also handled by this module. However, the BioFlow supports only centralized execution of workflows. Users need to learn BioFlow’s query language. The inter-program interactions and data conversions need be specified explicitly.

Triana Problem Solving Environment (PSE) (<http://www.trianacode.org/>) is another framework that supports graphical composition and distributed execution of web service

based scientific workflows [Maj04]. It supports dynamic services discovery, GUI based workflow composition, and distributed execution of workflows. Currently, the workflow composition is mapped to WS-BPEL code and can be executed on Triana Task Controllers (TSC). TSC's can be deployed on Grid based middleware platforms. One of the interesting features of the Triana workbench is the data type conversion tool. It can be dragged onto the canvas and connected among participation web services. However, more customized data type conversions such as extracting specific fields from an output of a service before feeding it to another need be programmed or manually performed by the user. While Triana provides a generic web service based platform for scientific workflows, Taverna (<http://taverna.sourceforge.net/>) provides a web service based platform for integrating data sources and tools in molecular biology [Hul06]. It has a comprehensive graphical user interface to compose and execute workflows. Unlike many other systems, Taverna clients need less system resources and computing power (personal computer). However, the system is very complex to learn. Taverna is build for Grid services, but can be extended to non-Grid based services.

Pegasus (<http://pegasus.isi.edu/>) is another comprehensive system for scientific workflows over Grid (and Web) services [Pegasus]. The Pegasus also provides a GUI based workflow composition. However, the Pegasus considers more on resource allocation and workflow task scheduling for large, long running Grid based workflows. GUI based workflow composition platform allows scientists to specify the workflow in the abstract level. Then, at the execution-level, the abstract level workflow is mapped onto more concrete workflow by specifying tasks to be executed, resource needed, and possible scheduling. It also, supports partial scheduling. Activities that are likely to be

executed in near future are scheduled to optimize resources. Data transfers among activities take place using GridFTP protocol. However, data transformations need be done manually or programmatically by the workflow developer. Unlike, Taverna, Pegasus is a bulky system and need considerable amount of expertise to develop applications. Many other web based tools for biological data analysis including MatchMiner [MatchMiner] from NIH, BioJava [BioJava], Bio-Perl [BioPerl] and GenePath [GenePath] exists.

These systems provide advanced features and almost all the platforms have graphical user interfaces to configure workflows. However, specific data transformations and conversions need to be done manually or programmatically by the developer. Moreover, these platforms require user to install systems and configure them before using the system. Also, accessibility is low in the sense that specially configured machines are needed. In addition, handheld based coordination of scientific workflow has not been supported or considered in any of these platforms. Handheld based monitoring will be very useful and increase the accessibility. We envision a platform which is available via web that allows scientists to configure, execute, and monitor their workflows with minimum effort. Thus, we believe that integrating different resources as per application requirement on the fly is still a distinct goal to achieve.

Section 8.2 presents few biological workflow examples that further illustrates requirements and issues. Then, Section 8.3 discusses how we can use the BondFlow system for biological workflows its strengths and weaknesses.

8.2 Motivating Example

Here we will discuss two biological workflow examples to illustrate the issues in the domain. More such workflows can be found at [DDBJ].

Alignment Region Comparison Workflow: Figure 8.1 show a workflow developed by (DNA Database of Japan) DDBJ to compare the alignment regions of high similar sequences of a given DNA sequence [DDBJ]. Alignment of gene sequences reflects the evolutionary relationship among genes. In a gene, genetic information is encoded using four letters, A, G, C, T. RiboNucleicAcid (RNA) is a nucleic acid generated from coding regions of a gene for further analysis. One of the common formats of representing DNA's and RNA's is FASTA format. FASTA file starts from the symbol ">" followed by a descriptive sequence of special identifies such as accession number. Rest of the file consists of gene coding sequence. For this workflow, first we input the gene sequence (RNA) of a " " in FASTA format.

Here, we use three different biological tools namely, BLAST, GetEntry, and ClustalW. First, we explain the functionality theses tools briefly and then we illustrate the operation of the workflow in detail.

BLAST (Basic Local Alignment Search Tool): The BLAST provides methods for searching of nucleotide and protein databases. Blast algorithm detects local as well as global sequential alignment regions of similarity embedded in otherwise unrelated proteins [DDBJ]. Sequence alignments provide a way to compare novel sequences with previously characterized genes. Both functional and evolutionary information can be inferred from well designed queries and alignments. The BLAST consists of about

twenty NCBI databases and six search programs [DDBJ]. The application developer needs to select a suitable program and a database to search from. It accepts several data formats as input such as FASTA.

GetEntry: GetEntry is another search tool developed by DDBJ (DNA Data Bank of Japan). It supports several Protein and DNA data sources that contain experimentally collected data. Users have to make a query based on one of the ID's such as Accession and Gene Name.

ClustalW: ClustalW is a general purpose multiple sequence alignment program for DNA or proteins. It produces biologically meaningful multiple sequence alignments of divergent sequences. ClustalW calculates the best match for the selected sequences, and lines them up so that the identities, similarities and differences can be seen. Evolutionary relationships can be seen. It supports about fifteen input formats and five output formats.

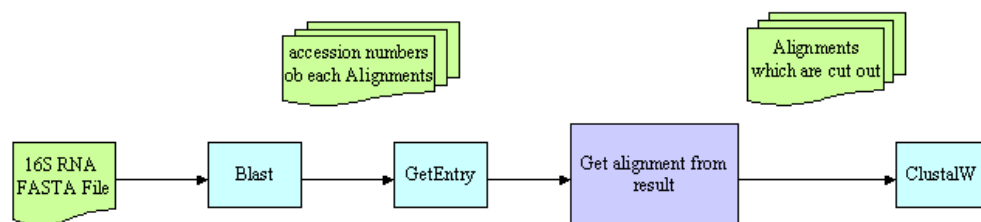


Figure 8.1: Alignment Region Comparison Workflow [DDBJ]

Operation of the Workflow: First, a BLAST query is made with FASTA file consists of 16S RNA. As we have mentioned earlier RiboNucleicAcid (RNA) is a nucleic acid

generated from coding regions of a gene for further analysis. One of the common formats of representing DNA's and RNA's is FASTA format. The BLAST query is made to the “ddbjct” database using blastn program. Blast query result consists of accession numbers, beginning and ending cordons of similar sequences. Next step of the workflow is to get alignments from the results. For that, we extract the accession numbers from the BLAST query and feed them in to the GetEntry service. GetEntry service retrieves entries from DDBJ database that retrieves actual sequence of similar sequences in FASTA format. Then the output of the GetEntry will be sent to ClustalW for multiple sequence alignment. Finally, similar sequences are matched using ClustalW service. Note that all the web services and data sources for this workflow are provided by DDBJ. DDBJ provides a Java application that implements this workflow. Here we presents the summary of the effort required to develop above workflow using Java (or similar programming language)

Estimated Development effort for non computer scientists (Java):

Total number of program files - 6

Line of Code Written - 407 lines (150 is reusable)

Estimated time - 1 month (non computer scientist)

Coordination - Centralized

Execution time - ~ 4 minutes

The next section demonstrates the implementation of the same workflow using the BondFlow system and similar evaluation has been made.

The above scenario demonstrates a typical set of operations or functions involving data and tool integration that the biologists deal with during the processing Gene analysis. In Section 8.3, we will revisit this scenario and illustrate how they can be accomplished using our BondFlow framework. Note that this scenario is “showcase” application for our infrastructure; the BondFlow system is not limited to only these scenarios but is capable of supporting a wide range of workflows for biological applications provided data sources and tools have web service interface.

8.3 Using the BondFlow System for Biological Workflows

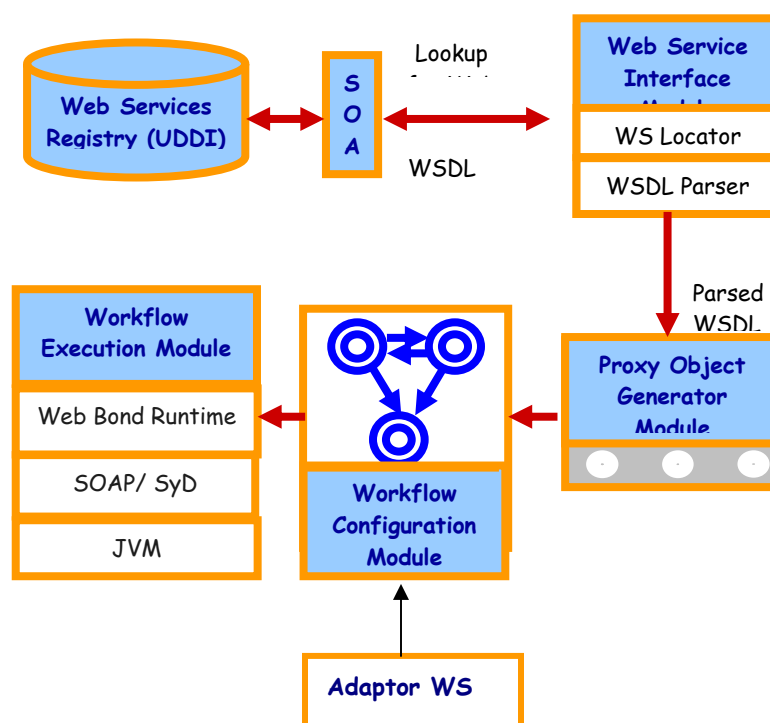


Figure 8.2: The BondFlow system for Biological Workflows

The BondFlow system provides an environment for configuring and executing workflows on the fly over heterogeneous web objects including web services [10]. We are planning to add two more components into our BondFlow system to facilitate biological workflows. First, an adaptor module that converts biological tools and data sources into web services. Second, a data adaptor web service that allows data conversion and transfer among biological tools and data sources. Currently, these components are in experimental stage.

Here, we exhibit development and deployment of the Alignment Region Comparison Workflow using the BondFlow system. We further demonstrate how such simple biological workflows can be created using this system using a stepwise methodology. This involves (a) finding biological data sources and tools, and wrapping them into web services; (b) generating data adaptor web services for each connector edge in the ad-hoc workflow; (c) configure the workflow over web-enabled tools, data sources, and data adaptors; (d) execute workflow. The current status of BondFlow system is as follows: Step (b) It automatically generates data adaptor code if the input-output regular expressions are specified, or if the data field selection and their permutation, if any, is specified. Step (c) It allows configuring preliminary biological workflows by selecting suitable web services and bonding them using our “web coordination bond” technology to enforce data and control dependencies [Bal05, Har04]. The conventional web services lack any bonding capabilities. Our system automatically generates coordinator proxy objects to web-bond-enable them [Bal05a]. The footprints of the wrappers are small enough to reside and executed on even on iPAQs. These bonding primitives are high-level specifications. Step (d) The BondFlow system allows execution and coordination

of configured workflows, even if the individual data sources or tools are located in a distributed fashion. The overhead introduced by the coordinator objects and by web bonds are only a small percentage of the total execution time on a typical workflow. Monitoring is currently limited to interacting with each workflow node individually. Step (a) on converting a tool into a web service is being addressed by many vendors and research groups, including, DDBJ, Microsoft (.net), and IBM (Websphere).

8.3.1 Workflow Development Methodology

Here, we explain the workflow configuration and execution using the BondFlow system.

Step 1, Selecting suitable data sources: Users of the BondFlow system initiate the workflow configuration by selecting suitable data sources/tools (Figure 8.4). The WSDL Parser parses the WSDL and allows the service components to be viewed in the form of summary of methods and parameter lists. Users can choose to save the viewed services for future reference.

Step2, Generate data adaptors and coordinator proxy objects: Once suitable data sources have been selected, users need to specify the data exchange requirements of data sources. This can be done as either input-output regular expression or can be directed to a program module to handle data transformation requirements. At this time, the system automatically generates data adaptors and the web service interface is created. The system also generates web bond enabled coordinator proxy objects (java object) for all selected data sources and adaptors [Bal05a].

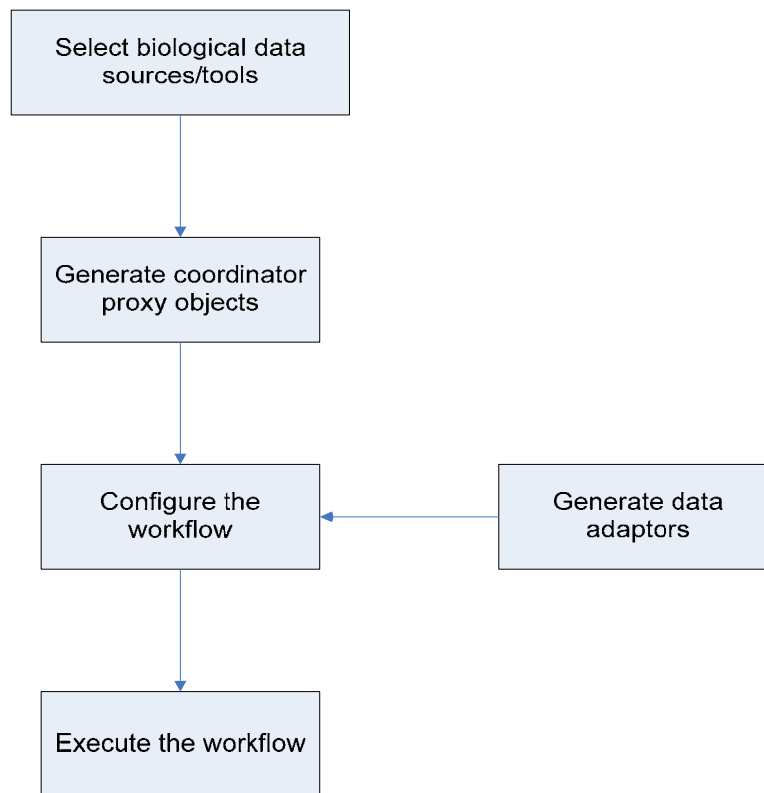


Figure 8.3: The BondFlow System: Users Perspective

Step 3, Configuring the workflow: Users of the system create web coordination bonds among the chosen services (now each data source/tool has a web service view) at any point of time to reflect data and control flow (using ``Subscription bonds"), and other dependencies (using ``Negotiation bonds"). Bond creation is done by the user selecting two data sources to be bonded and then specifying the bond type. The most important information provided at the bond creation time is the type of the bond to be created. Dependency enforcement and entire operation of bond execution depends on the type of the bond that has been created. Bond related information is stored in an XML storage file. The coordinator proxy object encompasses all the dependency modeling capabilities of

the web bond artifacts. Each web service method call is encapsulated by negotiation and subscription bond check. This logic makes sure that data and control dependencies are met before and after making the actual WS invocation [Bal05].

Step 4, Deployment of the workflow: Once any of the wrappers is invoked, the presence of the web bond is initially checked and depending upon the presence and type of the bond, coordination among components is carried out by enforcing the specified constraints and dependencies. Subscription bonds are used to transfer data from once data source/tool to another based on the constraints issue defined by the user.

8.3.2 Alignment Region Comparison Workflow using the BondFlow System

Here, we demonstrate our methodology for on-the-fly integration of the DNA alignment region comparison workflow. For this workflow, we have not used the data adaptor web service and data conversion has been accomplished manually. As shown in Figure 3, this is a simple workflow and needs to enforce only a sequential control flow dependencies. The BLAST and GetEntry web services have subscription bonds to GetEntry and ClustalW respectively. These subscription bonds make sure that data and control transfer. Data conversions are attached to subscription bonds. For example, we need to extract accession numbers from the Blast query results and feed them into the GetEntry web service. Currently these conversions are done using a Java program. However, we are extending our adaptor web service so that it handles automatic data conversion.

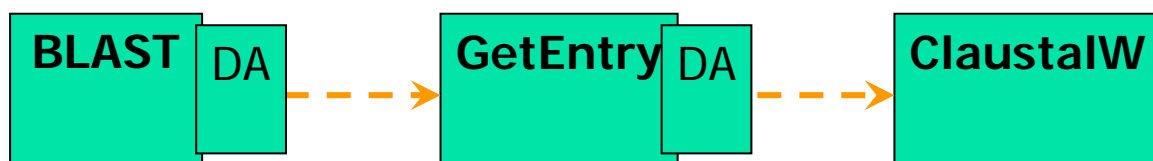


Figure 8.4 : Alignment Region Comparison Workflow using web coordination bonds

We have modeled this web service using our BondFlow system successfully.

Estimated Development effort using the BondFlow system:

Total number of program files - 3

Code Written – 142 lines for data conversion

Estimated time - 2 Weeks

Estimated if DA is available – few hours/few days

Execution time – ~ 4 min (~400 ms bond related)

Above figures clearly indicated that the BondFlow system provides platform that supports rapid application development platform. Once our data adaptor become functioning most of the data conversion requirements can also be automated providing more capable and easy use platform to develop and deploy such workflows.

8.3.3 System Output

This section walks through the execution of the workflow using the BondFlow system.

The Screen Shot of the current menu driven system: Figure 8.6 shows the starting point of the workflow. The query is a FASTA file to the BLAST web service

```

ClustalWSoap.class      InitValues.properties  ViewBond.java          runtemp.java
CreateBond.class        Menu.class             ViewWS.class           temp
CreateBond.java         Menu.java              ViewWS.java            test/
CurrencyExchangeService.class  ObjectData.properties  WSSStorage.xml         test.txt
CurrencyExchangeService.java  ParseMsdL.class        With Docum/            urls.txt
DeleteBond.class         ParseMsdL.java         Wrapper.class           version1_WithTimings/
DeleteBond.java          RunExternal.class      Wrapper.java            version1_WithTimings.zip

[~/Code/ThesisCode][10:17am] java runtemp test.txt
Find: test.txt
Query:
>AACY01004374.1
taattgaagatttgcacgtcagattgaacgtggcggttagcttaacacatgcaa
gtcgtgcgagaaagtatcttcggtatgagtagagcggcgaggggtgagtaacgcgtag
gaatctacgtagtagaagggatagcccggggaactcggattaatacgtatacctcct
ttgggagaagaagcctctcttgaagcttgcgtactagatgagcgtggttaagatta
gcttgggtgaggttaaaggtccacaaagcgaacgatctttagcgggtctgagaagcaga
cccgcaactgggactgagacacggcccgagactcctacgggaagcagcagtgagggaata
ttggcaaatggggcgaagcctgacccagcgaatcccggtgtgtgaagaagccctagggt
tgtaagcactttaagttgggaagaaggtattgtttaatacccaataacccgtgacatt
accttcagaataaagaccggcttaattcgtgcacagcagccgggttaacgggaaggtcca
agcgttaactggaattactggcggtaaagcgcgcgtaggtggtttattaaagttgatgtg
aaatccccgggtcacaactgggaactgcacccaactgattactagatagacgtagag
ggaggtagaattcacagtgtagcgttggaatgcgtagatattgtgaagaataccaatggc
gaaggcagcctcctggtatctgtactgacactaaagtgccgaagcgtgggtagcgaacagg
attagataccctggtagtccacgcgcgttaaacgatgcaactagctgttgggaacaatgt
tttccagtgccgcagctaacgtagtaagttgacgcctcgggaggtacggccgaaggtta
aaactcaaatgaattgacggggacccgcacaaagcgtggagcatgtgttttaattcgatg
caacgggaaaaacttaccatacttgacatacttggaaatcgttgaatgaagcagtgcc
cctcggggagccaagatacaggtgctgcacgtgctgtcagcgtgtgtctgtgagatgttc
cgttaagtcggataacgagcgaaccccttacccttatttgcagcaggttcggtcgggaac
tataggggactgcgggtgataaacccggggaaggtgaggaagcgcgtcaagtcacatgg
cccttaagtagaggtacacacgtgtacaaatggggatacacagacgggaacgtgaagcgc
gaggtggtgataactagaaaattctcgtagtcgggattggagtcgtgcaactcgtactc
catgaagtcggaatcgtagtaaatcgcgaatcagcatgtcgcgggtgaatcgttctcggg
tcttgatcacaccccgccgtataccatggaagtgattgcacacagaagtagatagcttaa
ccttcggggagggcgtttaccacggtgtgtctcatgactggggaagtcgtaaacaggtta
gccgtagggaacctgtggtgacacacctcta
Execution time for checking bonds : 86

```

Figure 8.5: The BondFlow System Executing Alignment Region Comparison Workflow

Step 1: Invoke Blast web service

Input to Blast: FASTA file consists of 16S RNA of a Gene sequence. As we have mentioned earlier RNA is a nucleic acid generated from coding regions of a gene for further analysis.

```

[~/Code/ThesisCode][10:17am] java runtemp test.txt
Find: test.txt
Query:
>AACY01004374.1

```

```

taattgaagagtttgatcatggctcagattgaacgctggcggtaggcttaacacatgcaa
gtcgtgcgagaaagtatcttcggatatgagtagcggcgacgggtgagtaacgcgtag
gaatctacctagtagaaggggtagcccggggaaactcggattaataccgtatacctcct
ttgggagaaagaaggcctctctttgaagcttgcgtactagatgagcctgcgtaagatta

```

Execution time for checking bonds : 86

Blast Result: Blast query result consists of accession numbers, beginning and ending cordons of similar sequences. For example, for the following output, accession number is AB212806 and the beginning and ending cordon positions are 190 and 949 respectively.

AACY01004374.1	AB212806 AB212806.1	89.47	760	80	0	220
979	190 949 0.0	872				

Step 2: Invoke the GetEntry web service

Input to the GetEntry:

Input to the GetEntry services is accession numbers and beginning and ending cordon positions in the sequence. For example, for the above sequence, the GetEntry search query will be AB212806 190 949. That fetches the gene sequence and other annotated data from the GetEntry database.

GetEntry Output:

```

id:AF468388calling ws
Return Value:LOCUS   AF468388           1436 bp   DNA   linear   BCT 06-NOV-
2003
DEFINITION   Arctic sea ice bacterium ARK10038 16S ribosomal RNA gene, partial
sequence.
ACCESSION   AF468388 VERSION   AF468388.1 KEYWORDS   .
SOURCE      Arctic sea ice bacterium ARK10038
ORGANISM    Arctic sea ice bacterium ARK10038
             Bacteria; Proteobacteria; Gammaproteobacteria; Pseudomonadales;
             Pseudomonadaceae; Pseudomonas.
REFERENCE   1 (bases 1 to 1436)
AUTHORS     Brinkmeyer,R., Knittel,K., Jurgens,J., Weyland,H., Amann,R. and
Helmke,E.
TITLE       Diversity and Structure of Bacterial Communities in Arctic versus

```

Antarctic Pack Ice
 JOURNAL Appl. Environ. Microbiol. 69 (11), 6610-6619 (2003)
 PUBMED 14602620
 REFERENCE 2 (bases 1 to 1436)
 AUTHORS Brinkmeyer,R. and Helmke,E.
 TITLE Direct Submission
 JOURNAL Submitted (15-JAN-2002) Pelagic Oceanography,
 Alfred-Wegener-Institut fuer Polar und Meeresforschung, Am
 Handelshafen 12, Bremerhaven D-27570, Germany
 FEATURES Location/Qualifiers
 source 1..1436
 /organism="Arctic sea ice bacterium ARK10038"
 /mol_type="genomic DNA"
 /isolate="ARK10038"
 /isolation_source="Arctic sea ice-melt pond"
 /db_xref="taxon:196822"
 rRNA <1..>1436
 /product="16S ribosomal RNA"
 BASE COUNT 354 a 323 c 464 g 293 t
 ORIGIN

```

1 atgcagtcag cgcgaaaggc cttcgggtg agtagagcgg cggacgggtg agtaacgcgt
61 aggaatctac ctggtagtgg gggataactt ggggaaactc aagctaatac cgcatacgcc
121 ctaaggggga aagcggggga tcttcggacc tcgcgtatt ggatgagcct gcgtaggatt

```

Step 3: Invoke the ClustalW web service

Input to ClustalW:

Input to the ClustalW is the concatenated result from the GetEntry database. This concatenated result will be used by the ClustalW for multiple sequence analysis.

```

>AB212806|g_proteobacterium_NEP68
gatgagcctgcgtaggattagcttgttggtgaggtaaaggctcaccaaggcgacgacatccttagctggtctgagaggatgatcag
ccacactgggactgagacacggcccagactcctacgggaggcagcagtggggaatattgcgcaatgggcgaaagcctgacg
cagccatgccgcgtgtgtgaagaaggccttcgggttgtaaagcactttcaattgggaagaaagggtgtacgttaatagcgtgcaa
ctgtgacgttacctttagaagaagcaccggctaactccgtgccagcagccggttaatacggagggtgcgagcgttaatcgga
attactgggcgtaaagcgcgcgtaggcgggttgtaagtcggatgtgaaagccctgggctcaacctgggaactgcattcgatact
ggccgactagagtacgagagagggaggtagaattccacgtgtagcggtgaaatgcgtagatatgtggaggaataaccggtggc
gaaggcggcctcctggctcgatactgacgctgaggtgcgaaagcgtgggtagcaaacaggattagataccctggtagtccacg
ccgtaaacgatgtctactagccgtgggagacttgattcttagtggcgcagctaacgcactaagtagaccgcctggggagtacg
gccgcaaggttaaaactcaaatgaattgacggggcccgacacaagcgggtggagcatgtggttaattcgatgcaacgcgaaga
accttacc>AY028196|m_bacterium_Tw-1

```

ClustalW Output:

CLUSTAL W (1.83) Multiple Sequence Alignments

Sequence format is Pearson

Sequence 1: AACY01004374.1 1535 bp

Sequence 2: AB212806|g_proteobacterium_NEP 759 bp

Start of Pairwise alignments

Aligning...

Sequences (1:2) Aligned. Score: 89

Guide tree file created: [/disk/xddbj/socket/clustalw/data/20060808233550908.dnd]

Start of Multiple Alignment

There are 1 groups

Aligning...

Group 1: Sequences: 2 Score:13347

Alignment Score 4719

CLUSTAL-Alignment file created

[/disk/xddbj/socket/clustalw/data/20060808233550908.aln]

20060808233550908.aln result

CLUSTAL W (1.83) multiple sequence alignment

AACY01004374.1

TAATTGAAGAGTTTGATCATGGCTCAGATTGAACGCTGGCGGTAGGCTTA

AB212806|g_proteobacterium_NEP -----

Finally the above results shows the alignment For example, for the above sequence with accession number AB212806, the alignment Score:13347 and the results is stores in the /disk/xddbj/socket/clustalw/data/20060808233550908.aln file.

8.4 Conclusions and Future Work

A large amount of biological data sources and tools are available for various data analysis purposes. However, a single tool or a data store could not serve all the requirements for myriad data analysis requirements (~ 1 billion databases). Thus, these tools and data sources need to be integrated in different ways. Among different approaches of data and

tool integration, web services provide better interoperability and scalability needed. Many efforts are already underway to convert these tools and data sources into web services. In this Chapter we have explored the usability of our BondFlow system as a platform for designing and executing biological workflows. We have successfully developed and deployed the Alignment Region Comparison Workflow using the BondFlow system. The development effort is significantly small and these workflows can be deployed on handheld devices giving more flexibility to users.

Currently our system is preliminary. In the future, we plan to integrate an automatic service adaptor that converts data sources and tools into web services on the fly. Also, we plan to extend the functionality of our data adaptor web service so that it supports various data conversions. Finally, we would like to publish our tool as a web based workflow development platform so that developers can configure their workflows and execute them on the web.

CHAPTER 9

CONCLUSIONS AND FUTURE WORK

Next generation Internet applications will be various kinds of collaborative applications among heterogeneous, autonomous entities (Objects). There is a definite trend in both industry and academia in adopting web based tools and technologies. Emergence of web services made this process more attractive for both communities. Web services solve the system and network heterogeneity issues. Such developments are transforming the web from information repository to a huge distributed computational platform. Thus, developing collaborative applications over Web has become increasingly important. Therefore, finding methodologies to rapidly develop and deploy robust collaborative applications is required.

Web services are software services distributed across the network. Users develop applications by integrating these software services into composite applications using appropriate coordination techniques. However, the current status of web service (Object) coordination and composition is a frenzied effort by many to shell out myriad of ever-richer protocols and languages for web service collaboration, suitable only for domain experts, without a substantial fundamental theoretical framework. Also, web services are stateless, passive entities in such composite applications requiring a centralized coordinator process. This makes such application development a tedious task. Also, this solution is less scalable and tightly coupled, which is not desirable for WWW applications. Therefore, in this dissertation we undertook the challenge of exploring (i) a fundamental set of bonding artifacts for composing web services, which are necessary

and sufficient in expressiveness and semantics, (ii) enhancing web service infrastructure to easily employ those core artifacts, and (iii) architecting a development and deployment platform to configure web service applications.

9.1 A Platform to Configure and Deploy Distributed Workflows over Web Services

This dissertation yields several significant results:

1. *Web Coordination Bonds*: Web coordination bonds allow applications to establish bonds among themselves to enforce dependencies. There are two types of web bonds: *subscription bonds* and *negotiation bonds*. The subscription bond allows automatic flow of information from a source entity to other entities that subscribe to it. This can be employed for synchronization as well as for more complex changes, needing data, control, or event flows. Negotiation bonds enforce dependencies and constraints across entities and trigger changes based on constraint satisfaction. Web bond primitives have sufficient modeling and expressive capabilities to enforce workflow dependencies; a feat none of the current dependency modeling technologies could accomplish comprehensively.

2. *Web Service Coordination Management Middleware (WSCMM)*: The WSCMM system transforms the current web services into state aware self-coordination entities. We accomplish this transformation by generating an “intelligent” web service coordinator proxy object (CPO) that represents a web service. These coordinator objects are stateful and they encapsulate all the capabilities of web coordination bonds enabling us to distribute workflow coordination among

participant web services (Figure 8.1). We have simulated our middleware architecture using the DEVS java simulation tool. Simulation results show that the middleware components behave accurately while enforcing workflow dependencies.

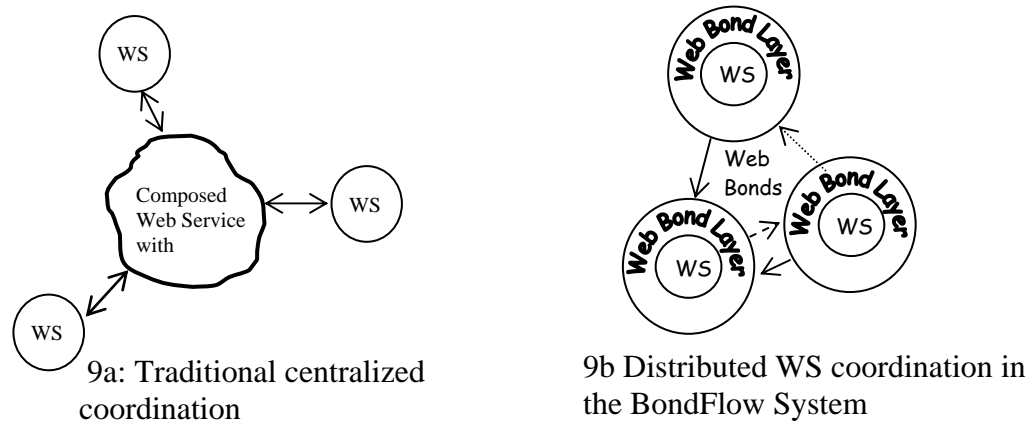


Figure 9.1: Workflow Coordination Architectures of the BondFlow System

2. *The BondFlow System:* The Bondflow system is based on web coordination bonds and our middleware platform. BondFlow is an easy to use platform to configure and execute distributed workflows over web services.

9.2 Future Work

Web coordination bonds are a set of capable coordination primitives. We strongly believe that these concepts have the formalism and rigor to become a “theory” for distributed coordination. It is worthwhile expanding this research further towards finding a theory for distributed coordination. We believe that the development of such a theory should

proved in parallel with the classification of different dependency patterns. In this dissertation we have investigated only control flow patterns and distributed communication patterns.

Another aspect of distributed coordination is enforcing QoS requirements. It will be a valid research effort to investigate how to enforce QoS requirements using web coordination bonds. It is highly likely that subscription bond has sufficient capabilities to help in specifying and enforcing QoS requirements.

Biological data and tool integration is one of the emerging research areas where web services will have a major impact. These data sources are heterogeneous (data types, data models, implementation technologies) in nature and the web service infrastructure is an ideal platform to hide this heterogeneity. Typically, non-computer scientists would prefer to compose their workflows (for any application in that matter) easily. Thus, the web is a very attractive environment for them. Extending the BondFlow system as a web-based tool to configure and execute biological (scientific) workflows is a very worthwhile endeavor. Our preliminary work in this area made us believe that the BondFlow system has sufficient capabilities to handle such applications.

BIBLIOGRAPHY

- [Aal05] W.M.P. van der Aalst, "Pi Calculus Versus Petri Nets: Let Us Eat Humble Pie Rather Than Further Inflate the Pi Hype," *BPTrends*, Vol. 3 No. 5, pp. 1-11, May 2005
- [Aal04] W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede, "Design and Implementation of the YAWL system," *Proc. of The 16th Intel. Conf. on Advanced Information Systems Engineering (CAiSE 04)*, Riga, Latvia, June 2004.
- [Aal03a] W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede, "Web Service Composition Languages: Old Wine in New Bottles?," *Proc. of the 29th EUROMICRO Conference: New Waves in System Architecture*, pp. 8-305. Los Alamitos, CA, 2003.
- [Aal03b] W. M. P. van der Aalst, "Don't go with the flow: Web services composition standards exposed," *TIEEE Intelligent Systems*, vol. 18, No. 1, pp. 72-79, Jan/Feb 2003.
- [Aal03c] Wil van der Aalst, Workflow patterns, <http://tmitwww.tm.tue.nl/research/patterns>, 2003
- [Atl04] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, S. Mock., Kepler: An Extensible System for Design and Execution of Scientific Workflows, *16th International Conference on Scientific and Statistical Database Management (SSDBM'04)*, 21-23 June 2004, Santorini Island, Greece
- [Att99]]MitBASE: a comprehensive and integrated mitochondrial DNA database. Attimonelli M, Altamura N, Benne R, ... et al. *Nucleic Acids Res.* 1999 Jan 1;27(1):128-33.
- [Aal02] W.M.P. van der Aalst and A.H.M. ter Hofstede, "Workflow Patterns: On the Expressive Power of (Petri net-based) Workflow Languages," *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2002)*, vol. 560 of DAIMI, pp. 1–20, Aarhus, Denmark, August 2002.
- [Aal02b] **Pattern-Based Analysis of BPML (and WSCI)**. W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and P. Wohed. QUT Technical report, FIT-TR-2002-05, Queensland University of Technology, Brisbane, 2002.
- [Aal98] W. M. P. van der Aals, "The Application of Petri Nets to Workflow Management," *Journal of Circuits, Systems and Computers*, Vol. 8, No. 1, 1998, pp. 21-66.

- [Ard03] L. Ardissono, A. Goy, G. Petrone, "Enabling conversations with web services," Proc. of the second international joint conference on Autonomous agents and multiagent systems, 2003, Melbourne, Australia July 14 - 18, 2003, Pages: 819 – 826
- [Alo96] G. Alonso, D. Agrawal, A. E. Abbadi, M. Kamath, R. Gunthor, and C. Mohan, "Advanced transaction models in workflow contexts," *In Proc. 12th Intl. Conf. Data Engineering, New Orleans, February 1996*, IEEE Computer Society Press, pp. 574-583.
- [And01] L. Andrade, J. Fiadeiro, "Coordination: The evolutionary dimension," *In Proc. 8th Intl. Conf. Technology of Object-Oriented Languages and Systems (TOOLS'01) Europe 2001, Zürich, Switzerland, March 12-14, 2001*, IEEE Computer Society Press, pp. 136-147.
- [Age74] T. Agerwala, "A complete model for representing the coordination of asynchronous processes," *Hopkins Computer Research Report No. 32*, Computer Science Program, Johns Hopkins Univ., Baltimore, Md., 58 pp., July 1974.
- [Age73] T. Agerwala, M. Flynn, "Comments on capabilities, limitations and 'correctness' of Petri nets"; *Proc. of the First Annual Symp. on Computer Architecture*, pp.81-86, 1973.
- [Alo04] G. Alonso, F. Casati, H. Kuno, V. Machiraju, "Web Services Concepts, Architectures and Applications Series Data-Centric Systems and Applications," Springer, ISBN: 3-540-44008-9, 2004.
- [Ark02] T. A. Arkin. TBusiness Process Modeling Language (BPML), Version 1.0T. *BPML.org*, 2002
- [Ave02] Lerina Aversano, Aniello Cimitile, Pierpaolo Gallucci, Maria Luisa Villan, "FlowManager: A Workflow Management System Based on Petri Nets," *In Proc. 26th Intl. Computer Software and Applications Conference (COMPSAC'02), August 26-29, 2002, Oxford, England*, IEEE Computer Society Press, pp. 1054-1059.
- [BioJava] BioJava <http://www.biojava.org/>
- [BioPerl] Bio-Perl, <http://bio.perl.org/>
- [Ben03] Benson D..A., Karsch-Mizrachi I., Lipman D.J., Ostell J., Wheeler D.L. GenBank. Nucleic Acids Res. 2003 Jan 1;31,1:23-7.
- [Ben00] Benson, D..A., Karsch-Mizrachi, I., Lipman, D.J., Ostell, J., Rapp, B.A. and Wheeler, D.L. Genbank. Nucleic Acids Res., 2000; 28, 1, 15-18.
- [Bry01] Ois Bry F., Kröger P. "A Molecular Biology Database Digest," Institute for Computer Science, University of Munich, Germany. <http://www.pms.informatik.uni-muenchen.de>, 2001

[Ben05] Boualem Benatallah, Fabio Casati, Daniela Grigori, H. R. Motahari Nezhad and Farouk Toumani. "Developing Adapters for Web Services Integration." *Procs of CAiSE 2005*. Porto, Portugal. Jun 2005.

[Bal05a] Janaka Balasooriya, Mohini Padye , Sushil Prasad, and Shamkant B. Navathe "BondFlow: A System for Distributed Coordination of Workflows over Web Services," *In 14Pth HCW* in conjunction with IPDPS 2005. Denver, Colorado, USA, April .

[Bal05b] Janaka Balasooriya and Sushil K. Prasad, "Toward Fundamental Primitives and Infrastructure Enhancements for Distributed Web Object Coordination and Workflows," *Proc. International Conf. on Web Services (ICWS'05)*, Orlando, July 2005.

[Bal05c] Janaka Balasooriya and Sushil K. Prasad, "A Middleware Architecture for Conversation-aware Stateful Web Services for Distributed Coordination," Tech Report, CS-TR-05-03, Georgia State University, July 2005, 33 pages. <http://www.cs.gsu.edu/~cscjlbx/wscmsm.pdf>

[Bar05] A. Barros, M. Dumas, and P. Oaks, "Standards for Web Service Choreography and Orchestration: Status and Perspectives," *Proceedings of the Workshop on Web Services Choreography and Orchestration for Business Process Management*, Nancy, France, September 2005.

[Bar05] A. Barros, M. Dumas, and P. Oaks. Standards for Web Service Choreography and Orchestration: Status and Perspectives. To appear in *Proc. of the Workshop on Web Services Choreography and Orchestration for Business Process Management*, Nancy, France, September 2005.

[Bru05] J. de Bruijn, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu, M. Kifer, D. Fensel. "The Web Service Modeling Language WSML," *WSML Deliverable D16.1v0.2*, 2005. <http://www.wsmo.org/TR/d16/d16.1/v0.2/>

[Bal05] Janaka Balasooriya, Mohini Padye , Sushil Prasad, and Shamkanth B. Navathe, "BondFlow: A System for Distributed Coordination of Workflows over Web Services," *Proc. 14th Heterogeneous computing workshop* in conjunction with IPDPS 2005. Denver, Colorado, USA, April 4.

[Bou05] Boualem Benatallah, Marlon Dumas, Quan Z. Sheng, "Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services," *Distributed and Parallel Databases* Vol. 17 No. 1, pp. 5-3, 2005.

[Ben03] B. Benatallah, P. Chrzastowski-Wachtel, R. Hamadi, M. O'Dell, and A. Susanto, "HiWorD: A Petri Net-based Hierarchical Workflow Designer," *Proc. of the 3rd Intl. Conf. on Application of Concurrency to System Design (ACSD 2003)*, IEEE Society, Guimaraes, Portugal, June 2003.

[Eli05] Elisa Bertino, Alessandro Provetti, and Franco Salvetti, “Reasoning about RDF statements with default rules.” Proc. of the Rule Languages for Interoperability Conf., Washington, April 2005.

[Ber03] Fran Berman, Geoffrey Fox and Tony Hey "Grid Computing: Making the Global Infrastructure a Reality " Wiley, 2003

[Ben02] B. Benatallah, M. Dumas, M. C. Fauvet and F. A. Rabhi, “Towards Patterns of Web Services Composition,” *In Patterns and Skeletons for Parallel and Distributed Computing*, Springer Verlag (UK), 2002.

[Bic03] M Bichler, G Kersten and S. Strecker, “Towards a Structured Design of Electronic Negotiations”, Group Decision and Negotiation, 2003, Vol. 12, No. 4, p. 311 - 335.

[Ben03] B. Benatallah, Q. Z. Sheng, and M. Dumas “*The SELF-SERV Environment for Web Services Composition*,” IEEE Internet Computing 7(1):40-48, January/February 2003. IEEE Computer Society.

[Bra03] Brahim Medjahed , Athman Bouguettaya , Ahmed K. Elmagarmid, “Composing Web services on the Semantic Web,” The VLDB Journal — The International Journal on Very Large Data Bases, v.12 n.4, p.333-351, November 2003.

[Bru05] Bruni, Hernán Melgratti, and Ugo Montanari, “Theoretical foundations for compensations in flow composition languages,” *Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, Vol. 40, Issue 1, January 2005.

[Bra03] Brahim Medjahed , Athman Bouguettaya , Ahmed K. Elmagarmid, “Com-posing Web services on the Semantic Web,” *The International Journal on Very Large Data Bases*, v.12 n.4, p.333-351, November 2003.

[Ben02] B. Benatallah, M. Dumas, M. C. Fauvet, F. A. Rabhi, and Quan Z. Sheng, “Overview of Some Patterns for Architecting and Managing Composite Web Services,” *ACM SIGecom Exchanges*, ACM Press, August 2002, pp. 9-1.

[Bru05] Theoretical foundations for compensations in flow composition languages Roberto Bruni, Hernán Melgratti, Ugo Montanari, ACM SIGPLAN Notices , Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages, Volume 40 Issue 1, January 2005 – Transactions.

[Bus05] N. Busi R. Gorrieri, C. Guidi, R. Lucchi, G. Zavattaro, “Towards a formal framework for Choreography,” *3rd International Workshop on Distributed and Mobile Collaboration (DMC 2005)*, IEEE Computer Society Press.

[Bus94] C. Bussler, and S. Jablonski, "Implementing agent coordination for workflow management systems using active database systems," *Proc. of 4Pth Intl. Workshop on TResearch Issues in Data Engineering: Active Database Systems*, T Houston, Texas, IEEE Computer Society Press, pp. 53–59, February 1994.

[Bou05] Boualem Benatallah, Fabio Casati, Daniela Grigori, H. R. Motahari Nezhad and Farouk Toumani. Developing Adapters for Web Services Integration. *Procs of CAiSE 2005*. Porto, Portugal. Jun 2005.

[Cas90] T.L. Casavant, J.G. Kuhl, "A Communicating Finite Automata Approach to Modeling Distributed Computation and its Application to Distributed Decision-Making," *IEEE trans. on computers*, May 1990, Vol. 39, No. 5, pp. 628-639.

[Cru02] A. J. A. Cruz, A. B. Raposo, L. P. Magalhaes, "Coordination in Collaborative Environments - A Global Approach," *7th Intel. Conf. on Computer Supported Cooperative Work in Design (CSCWD 2002)*, Rio de Janeiro, Brazil, 2002, pp. 25 – 30

[Chr03] Christoph Bussler, "Semantic Web Services: The Future of Integration!," *7Pth East-European Conf. on Advances in Databases and Information Systems (ADBIS' 03)*, Dresden, Germany, pp. 1-2, September 2003.

[Chr04] Christoph Schuler, Rogr Weber, Heiko Schuldt, Hans-Jörg Schek, "Scalable Peer-to-Peer Process Management - The OSIRIS Approach." *ICWS 2004*: 26-34

[Car99] L. Cardelli, and R. Davies, "Service Combinators for Web Computing," *IEEE Transactions on Software Engineering*, Vol. 25, No. 3, pp. 309-316, 1999.

[Cha04] Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha, "Service Composition for Mobile Environments," *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, February, 2004

[Cur04] Val Curwen et al. The Ensembl Automatic Gene Annotation System. *Genome Res.* 2004 14: 942-950.

[Cza04] K. Czajkowski., Ferguson, D., Foster, I ...et al. The WS-Resource Framework. <http://www-106.ibm.com/developerworks /library/ws-resource/ws-wsrf.pdf> , 2004

[Cha04] Dipanjan Chakraborty, Anupam Joshi, Tim Finin, and Yelena Yesha, "Service Composition for Mobile Environments," *Journal on Mobile Networking and Applications, Special Issue on Mobile Services*, February, 2004

[Gir04] Girish Chafle, Sunil Chandra, Vijay Mann and Mangala G. Nanda, "Decentralized Orchestration of Composite Web ServicesTTH," T In *TProceedings of the Alternate Track on Web Services at the 13th International World Wide Web Conference(WWW 2004)*T, New York, NY, May 2004.

[Coo05] Dominic Cooney, Marlon Dumas, and Paul Roe “A Programming Language for Web Service Development” Twenty-Eighth Australasian Computer Science Conference (ACSC 2005) pp. 143-150, January 2005.

[Fra03] Francisco Curbera, Rania Khalaf, Nirmal Mukhi, Stefan Tai, Sanjiva Weerawarana, “Service-oriented computing: The next step in Web services,” *Communications of ACM*, Vol. 46 , No. 10, pp. 29 – 34, October 2003.

[Cza04] K. Czajkowski., Ferguson, D., Foster, I ...et al. The WS-Resource Framework. <http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf> , 2004

[Dat02] A. Datta and S. H. Son, “A study of concurrency control in real-time, active database systems,” *IEEE Trans. on Knowledge and Data Engineering*, Vol:14 Issue:3, IEEE Computer Society Press, June 2002, pp. 465-484.

[Dus04] Schahram Dustdar, Harald Gall, Roman Schmidt, “Web Services for Groupware in Distributed and Mobile Collaboration,” *PDP 2004*: 241-

[DiscoveryNet] <http://www.discovery-on-the.net/>

[DDBJ] <http://xml.nig.ac.jp/wsdl/index.jsp>

[DEVSTJava] DEVS simulator, <http://www.acims.arizona.edu/SOFTWARE>

[Dou03] L. Doug and V. Steve “Introduction - middleware for web services,” *IEEE Distributed Systems Online*, 1(4), 2003.

[Dou03] L. Doug and V. Steve, “Introduction - middleware for web services,” *IEEE Distributed Systems Online*, Vol. 1 No. 4, 2003.

[Dou03] Doug Lea and Steve Vinoski “Introduction - Middleware for Web Services,” *IEEE Distributed Systems Online* Vol. 4 No. 1 2003.

[Dus04] S. Dustdar, “Web Services Workflows - Composition, Coordination, and Transactions in Service-Oriented Computing,” *Concurrent Engineering: Research and Applications*, Sage Publications, p. 237-246, September 2004.

[ebXML02] ebXML,Business Process Specification Schema (Version 1.01). <http://www.ebxml.org/specs/ebBPSS.pdf>, May 11 2002.

[FAT02]]Analysis of the mouse transcriptome based on functional annotation of 60,770 full-length cDNAs. The FANTOM Consortium and the RIKEN Genome Exploration Research Group Phase I & II Team. *Nature* 420:563-573, 2002.

[Gri82] E. J. J. v. Griethuysen, "ISO- Concepts and Terminology for the Conceptual Schema and the Information Base", N695, ISO/TC9/SC5/WG3, 1982.

[Gor05] R. Gorrieri, C. Guidi and R. Lucchi, "Reasoning about interaction patterns in Choreography" In Proc. of 2nd International Workshop on Web Services and Formal Methods (WS-FM '05). Volume 3070 of LNCS, pages 333-348. 2005.

[Gua03] Zhijie Guan and Hasan M. Jamil, "Streamlining biological data analysis. using BioFlow," In Proc. of the 3rd IEEE Symposium on Bioinformatics and BioEngineering, 2003

[GenePath] GenePath, <http://www.genepath.org/>

[Gri01] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. Katz, Z. Mao, S. Ross, and B. Zhao, "The Ninja Architecture for Robust Internet-Scale Systems and Services," Computer Networks, Special Issue on Pervasive Computing, 2001.

[Gua98] Yang Guangxin, Shi Meilin, Xiang Yong, Wu Shangguang , "Wowww! :managing workflow on the World Wide Web," In Proc. Intl. Conf. Communication Technology (ICCT '98), Oct 22-24, 1998, IEEE Computer Society Press, pp. 1- 5.

[Guo02] Jiang Guo, "Using Category Theory to Model Software Component Dependencies," Proc. 9th Annual IEEE Intl. Conf. and Workshop on the Engineering of Computer-Based Systems (ECBS 2002) , April 08 - 11, 2002 ,Lund, Sweden, IEEE Computer Society Press, pp. 185-194.

[Gre02] Paul Grefen, "Transactional workflows or workflow transactions," In Proc. 13th Intel. Workshop on Database and Expert Systems Applications (DEXA 2002), September 02-06, 2002, Aix-en-Provence, France, LNCS 2453, Springer-Verlag Berlin Heidelberg, pp. 60-69.

[Hua98] S. H. S. Huang, "Building business processes using a state transition model on World Wide Web Application-Specific Software Engineering Technology," In Proc. 1st IEEE Workshop on Application-Specific Software Engineering and Technology (ASSET'98), Richardson, Texas, 26-28 Mar , 1998, IEEE Computer Society Press, pp. 2 -7

[Hof96] H. M. Arthur ter Hofstede, Maria E. Orlowska, Jayantha Rajapakse, "Verification Problems in Conceptual Workflow Specifications," Proc. of the 15th Intel. Conf. on Conceptual Modeling, (ER'96), Cottbus, Germany, October 1996 , pp. 73-88.

[Hul06] Duncan Hull, Katy Wolstencroft, Robert Stevens, Carole Goble, Matthew Pocock, Peter Li and Tom Oinn. Taverna: A tool for building and running workflows of services. Nucleic Acids Research :34 (Web Server Issue), July, 2006.

[Her04] Thomas Hernandez and Subbarao Kambhampati, *Integration of Bioinformatic Sources: Current Approaches and Systems* ASU CSE TR 03-005. July 2003. SIGMOD Record, Vol 33, No 3 September 2004.

[Hum05] M. Humphrey, G. Wasson, K. Jackson ... et al. "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations." *14th IEEE Intl. Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, NC, 24-27 July 2005

[Haw05] Hawryszkiewicz, I., Steele, R. "Extending Collaboration to Mobile Environments," In the Proceedings of the International Conference on Web Technologies, Applications and Services, Calgary, Canada, July 4-6, 2005.

[Haw05] Hawryszkiewicz, I., Steele, R. "Extending Collaboration to Mobile Environments," *In the Proc. of the International Conference on Web Technologies, Applications and Services*, Calgary, Canada, July 4-6, 2005.

[Har04]Arthi Hariharan, Sushil K. Prasad, Anu G. Bourgeois, Erdogan Dogdu, Sham Navathe, Raj Sunderraman, and Yi Pan", A Framework for Constraint-Based Collaborative Web Service Applications and a Travel Application Case Study " *Proc. of the 2004 International Symposium on Web Services and Applications (ISWS'04)* June 21-24, 2004, Las Vegas, Nevada, USA.

[Hum05] M. Humphrey, G. Wasson, K. Jackson ... et. al. "State and Events for Web Services: A Comparison of Five WS-Resource Framework and WS-Notification Implementations." *14th IEEE Intl. Symposium on High Performance Distributed Computing (HPDC-14)*, Research Triangle Park, NC, 24-27 July 2005

[Hua02] P. C. K. Huang. and Ji-Ye Mao, "Modeling e-Negotiation Activities with Petri Nets, Proceedings of the 35th Hawaii International Conference on System Sciences", *HICSS 35*. Hawaii. 2002.

[Hul04] R. Hull and J. Su. "Tools for Design of Composite Web Services," *ACM SIGMOD International Conference on Management of Data, June 2004* .

[Hul04] Richard Hull, Michael Benedikt, Vassilis Christophides, Jianwen Su, "E-Services: A Look Behind the Curtain," *Proc. of the twenty-second ACM SIGMOD-SIGACT-SIGART symp.on Principles of database systems*, San Diego, California, June 09 - 11, 2003, pp. 1-14.

[Jor05] Jørstad, I., Dustdar, S., van Do, T. "A Service Oriented Architecture Framework for collaborative services," *3rd International Workshop on Distributed and Mobile collaboration (DMC)*, IEEE WETICE, 13 - 15 June 2005, Linköping, Swe-den, IEEE Computer Society Press.

[Jon03] Johnson P Thomas, Mathews Thomas, George Ghinea, "Modeling of Web Services Flow," *IEEE Intl. Conf. on E-Commerce*, June 24 - 27, 2003 Newport Beach, California , pp. 331-339.

[Jen87] K. Jensen. Coloured Petri Nets. "Advances in Petri Nets," *Lecture Notes in Computer Science*. Springer-Verlag, volume 254-255, Berlin-New York, 1987.

[Jor05] Jørstad, I., Dustdar, S., van Do, T. "Service-Oriented Architectures and Mobile Services." *3rd Intl. Workshop on Ubiquitous Mobile Information and collaboration Systems (UMICS), co-located with CAiSE 2005*, 13 - 14 June 2005, Porto, Portugal.

[Kim02] Ki-Chan Kim and Il Im, "The Effects of Electronic Supply Chain Design (e-SCD) on Coordination and Knowledge Sharing: An Empirical Investigation", *Proc. of the 35th Hawaii International Conference on System Sciences*. HICSS 35. Hawaii. 2002.

[Kim02] Ki-Chan Kim and Il Im, "The Effects of Electronic Supply Chain Design (e-SCD) on Coordination and Knowledge Sharing: An Empirical Investigation", *Proceedings of the 35th Hawaii International Conference on System Sciences*. HICSS 35. Hawaii. 2002.

[Kie02] B. Kiepuszewski, "Expressiveness and Suitability of Languages for Control Flow Modeling Workflows," PhD thesis, *Queensland University of Technology*, Brisbane, Australia, 2002.

[Ko03] In-Young Ko, R. Neches, "Composing Web Services for Large-Scale Tasks," *Internet Computing, IEEE* , Vol.7 No. 5 , Sept.-Oct. 2003 , pp. 52 –59.

[Lak94] C. A. Lakos "Object Petri Nets — Definition and Relationship to Coloured Nets," *Technical Report R94-3, Department of Computer Science, University of Tasmania*, April 1994.

[Lak95] C. A. Lakos. "The Object Orientation of Object Petri Nets," *Proceedings of the first international workshop on Object-Oriented Programming and Models of Concurrency - 16th International Conference on Application and Theory of Petri Nets*, pages 1–14, June, 1995.

[Fra01] Frank Leymann, "Web Services Flow Language (WSFL 1.0)." *IBM Software Group*, May 22, 2001.

[Jan03] Julian Jang, Alan Feteke, Paul Greenfield, Dean Kuo "Expressiveness of Workflow Description Languages" *proc. of ICWS'03* , June 23 - 26, 2003, Las Vegas, Nevada, USA, pp. 104-110

[Ley02] F. Leymann, D. Roller, and M.-T. Schmidt, "Web services and business process management," *IBM systems Journal*, Vol 41, No 2, 2002.

[Lab] A. Labarga, M. Anderson, F. Valentin, R. Lopez WEB SERVICES AT THE EUROPEAN BIOINFORMATICS INSTITUTE European Bioinformatics Institute, Hinxton, United Kingdom.

[Lan03] D. Langworthy (ed.) *Web Services Coordination (WS-Coordination)*. Published online at <http://www-106.ibm.com/developerworks/library/ws-coor/>, 2003

[Lan03] D. Langworthy (ed.) *Web Services Atomic Transaction (WS-AtomicTransaction)*. Published online at <http://www-106.ibm.com/developerworks/library/ws-atomtran/>, 2003

[Lit03] Mark Little , “Service-oriented computing,” *Transactions and Web services, ACM*, Vol. 46 , No.10 October 2003, pp. 49 – 54

[Lom01] A.R Lomuscio, M. Wooldridge and N. R Jennings, “A classification scheme for negotiation in electronic commerce”. In: *Agent-Mediated Electronic Commerce: A European Agent Link Perspective*, 2001, p. 19-33.

[Lim02] J. Lim, B. Gan and Ting-Ting Chang, “A Survey on NSS Adoption Intention”, *Proceedings of the 35th Hawaii International Conference on System Sciences, HICSS 35. Hawaii.2002.*

[Ler02] Lerina Aversano, Aniello Cimitile, Pierpaolo Gallucci, Maria Luisa Villan, “FlowManager: A Workflow Management System Based on Petri Nets,” *In Proc. 26th Intl. Computer Software and Applications Conference (COMPSAC’02), August 26-29, 2002, Oxford, England, IEEE Computer Society Press* , pp. 1054-1059.

[Luc]R. Lucchi, and M. Mazzara. “A Foundational Mechanism for WS-BPEL Recovery Framework,” Paper submitted to *Journal of Logic and Algebraic Programming (JLAP)*, Elsevier press.

[Lam98] M.D. Lambert, C. M. Cooper and D. J. Pagh, ”Supply Chain Management: Implement Issues and Research Opportunities”, *International Journal of Management Logistics*. Vol 9, N 2.1998.

[Lip96] E. Lippe, Arthur H. M. ter Hofstede, “A Category Theory Approach to Conceptual Data Modeling”” *Informatique Theorique et Applications,(ITA)*, 1996, vol. 30 No. 1 pp. 79

[Luc] R. Lucchi, M. Mazzara. “A Foundational Mechanism for WS-BPEL Recovery Framework,” Paper submitted to *Journal of Logic and Algebraic Programming (JLAP)*, Elsevier press.

[Mna04] Adel Ben Mnaouer, Anand Shekhar, Zhao Yi-Liang, “A Generic Framework for Rapid Application Development of Mobile Web Services with Dynamic Workflow Management,” *IEEE SCC 2004*: 165-171

[Maj04] S. Majithia, M. S. Shields, I. J. Taylor, I. Wang: *Triana: A Graphical Web Service Composition and Execution Toolkit*. Proceedings of the IEEE International Conference on Web Services (ICWS'04), pp. 514-524. IEEE Computer Society, 2004

[Mna04] Adel Ben Mnaouer, Anand Shekhar, Zhao Yi-Liang, "A Generic Framework for Rapid Application Development of Mobile Web Services with Dynamic Workflow Management," *IEEE SCC* 2004: 165-171

[MatchMiner] MatchMiner, <http://discover.nci.nih.gov/matchminer/index.jsp>

[Mue05] zur Muehlen, Michael; Stohr, Edward A. "Internet-enabled Workflow Management," Editorial to the Special Issue of the Business Process Management Journal 11 (2005)

[MQs] "IBM MQSeries Workflow Concepts and Architecture Version 3. 1," IBM Danmark A/S GH12-6285-00 , July 1998.

[Mei96] J. Meidanis, G. Vossen, M. Weske, "Using workflow management in DNA sequencing," *In Proc. 1st IFCIS Intl. Conf. Cooperative Information Systems (CoopIS'96), June 19-21, 1996, Brussels, Belgium*, IEEE Computer Society Press, pp. 114-123.

[Men04] Mendling, J., Strembeck, M., Neumann, G.: Extending BPEL4WS for Multiple Instantiation. In Dadam, P., Reichert, M., eds.: *INFORMATIK 2004 - Band 2*, Proceedings of the 34th Annual Meeting of German Informatics Society (GI), GPA 2004. Volume 51 of Lecture Notes in Informatics (2004) 524--529

[Moh98a] C. Mohan, "Workflow Management in the Internet Age," *Proc. 2nd East-European Symp. on Advances in Databases and Information Systems (ADBIS'98)*, Poznan, Poland, September, 1998, LNCS, Vol. 1475, pp. 26-34.

[Moh98b] C. Mohan, "Recent Trends in Workflow Management Products, Standards and Research," *NATO ASI Series, Series F: Computer and Systems Sciences*, August 1998 ,Volume 164, Springer Verlag, pp. 396-409.

[Mal94] T. W. Malone, and K. Crowston, "The interdisciplinary study of coordination," *ACM Comput. Surv. Vol:26 Issue:1, 1994*, pp. 87-120.T

[MaL02] T. McLaren, M. Head and Y. Yuan, "Supply chain collaboration alternatives: understanding the expected costs and benefits," *Internet Research: Electronic Networking Applications and Policy*. 2002, V: 12, N 4.

[Mel04] M. Melise, Veiga de Paula, Jano M. de Souza, José R. Blaschek, Fernanda Baião,T "A Cooperative System to Support Inventory Leveling Negotiations," *HICSS* 2004, TPage: 70192.2

[Moo04] Aldo de Moor and Willem-Jan van den Heuvel, "Web Service Selection in Virtual Communities," *Proc. 37th Hawai'i International Conference on System Sciences*, Big Island, Hawaii, January 5-8, 2004, pp. 701-717.

[Mur89] T. Murata, "Petri nets: properties, analysis and applications"; *Proceedings of IEEE*, vol.77, no.4, pp.541-580, 1989.

[Nek03] Nektarios Gioldasis, Nektarios Moutzouris, Fotis G. Kazasis, Nikos Pappas, Stavros Christodoulakis, "A Service Oriented Architecture for Managing Operational Strategies," *ICWS-Europe*, pp. 11-23, Germany, September 2003

[Nav04] Shamkant B. Navathe, Upen Patil, "Genomic and Proteomic Databases and Applications: A Challenge for Database Technology," *DASFAA 2004*: 1-24.

[Nor99] Norman W. Paton, Oscar Diaz, "Active Database Systems," *ACM Computing Surveys*, Vol. 31, No. 1, March 1999, pp 63 - 103.

[Orr03] B. Orriens, J. Yang, and M.P. Papazoglou, "A Framework for Business Rule Driven Service Composition," *Proc. of the Fourth International Workshop on Conceptual Modeling Approaches for e-Business Dealing with Business Volatility*, Chicago, United States, October 13-16, 2003.

[Pap05] M. P. Papazoglou and W.J. van den Heuvel, "Web Services Management: A Survey," *IEEE Internet Computing* Nov/Dec. 2005

[Pegasus] Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems *Scientific Programming Journal*, January 2005.

[Pao05] Romano, Paolo; Marra, Domenico; Milanese, Luciano (2005) " Web services and workflow management for biological resources" *BMC Bioinformatics* 6 S24

[Pel03] Chris Peltz , "Web Services Orchestration and Choreography," *IEEE Computer*, Vol. 36, No. 10, October 2003, pp. 46-52.

[Pic99] G. Piccinelli, "TService Provision and Composition in Virtual Business Communities," *Tech. report HPL-1999-84*, Hewlett-Packard, TPalo Alto, Calif., T1999, <http://www.hpl.hp.com/techreports/1999/HPL-1999-84.html>.

[Pra05] Sushil K. Prasad and J. Balasooriya, 2005, "Fundamental Capabilities of Web Coordination Bonds: Modeling Petri Nets and Expressing Workflow and Communication Patterns over Web Services ", *Proc. Hawaii Intl. Conf. in Syst. Sc. (HICSS-38)*, Jan., Big Island, January 4-8.

[Pra04a] Sushil K. Prasad, V. Madiseti, et al. 2004. "System on Mobile Devices (SyD): A Middleware Testbed for Collaborative Applications over Small Heterogeneous

Devices and Data Stores,” *Procs. ACM/IFIP/USENIX 5th International Middleware Conference*, Toronto, Ontario, Canada, October 18th - 22nd.

[Pra04b] Sushil K. Prasad and Janaka Balasoorya, “Web Coordination Bonds: A Simple Enhancement to Web Services Infrastructure for Effective Collaboration,” *Proc. 37th Hawai’i International Conference on System Sciences*, Big Island, Hawaii, January 5-8, 2004, pp. 70192.1

[Pra04c] Sushil K. Prasad and J. Balasooriya. 2004. “Web Coordination Bonds: A Simple and Theoretically Sound Framework for Effective Collaboration among Web Services,” Technical Report CS-TR-04-01, *Department of Computer Science, Georgia State University*, June, 36 pages. <http://www.cs.gsu.edu/~cscjlbx/webond.pdf>

[Pra03a] Sushil K. Prasad, et al. “Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile Data Stores Using SyD Link Technology,” *In Proc. Mobile Wireless Network Workshop held in conjunction with The 23rd Intl. Conf. Distributed Computing Systems (ICDCS’03)*, May 19-22 2003, Providence, Rhode Island, IEEE Computer Society Press, pp. 803 –811.

[Pra03b] Sushil K. Prasad, et al. “Implementation of a Calendar Application Based on SyD Coordination Links,” *In Proc. 3Prd Intl. Workshop Internet Computing and E-Commerce in conjunction with the 17th Annual Intl. Parallel & Distributed Processing Symp. (IPDPS 2003)*, 22-26 April, Nice, France, IEEE Computer Society Press, pp. 242.

[Pre02] Gunter Preuner and Michael Schrefl, “Integration of Web Services into Workflows through Multi-Level Schema Architecture,” *In Proc. 4Pth IEEE Intl. workshop on Advanced issues of E-Commerce and Web-based Information Systems (WECWIS’02)*, 26–28 June ,2002 , Newport Beach, CA, IEEE Computer Society Press, pp. 51-60.

[Ros05] Rosenberg, F., Dustdar, S. “Towards a Distributed Service-Oriented Business Rules System,” *IEEE European Conference on Web Services (ECOWS)*, 14-16 November 2005, IEEE Computer Society Press

[Ran04] Anand Ranganathan , Scott McFaddin , “Using Workflows to Coordinate Web Services in Pervasive Computing Environments,” *Proc. of the IEEE International Conference on Web Services (ICWS’04)*, June 6-9, 2004, San Diego, California, USA. IEEE Computer Society 2004, 288-295

[Ran04] Anand Ranganathan , Scott McFaddin , “Using Workflows to Coordinate Web Services in Pervasive Computing Environments,” *Proc. of the IEEE International Conference on Web Services (ICWS’04)*, June 6-9, 2004, San Diego, California, USA. IEEE Computer Society 2004, 288-295

[Rap00] A. B. Raposo, L. P. Magalhaes, I. L. M. Ricarte, "Coordinating Activities in Collaborative Environments: A High Level Petri Nets Based Approach," *SCI 2000 - 4th World Muticonference on Systemics, Cybernetics and Informatics*, Orlando, pp. 195-200.

[Ros05] Rosenberg, F., Dustdar, S. "Towards a Distributed Service-Oriented Business Rules System," *IEEE European Conference on Web Services (ECOWS)*, 14-16 November 2005.

[Rap00] A. B. Rapso, L. P. Magalhaes, I. L. M. Ricarte, "Petri Nets Based Coordination Mechanisms for Multi-Workflow Environments," *International Journal of Computer Systems Science & Engineering*, September 2000, vol.15, no .5, p.315 - 326,.

[RDF04] RDF/XML Syntax Specification (Revised) W3C Recommendation 10 February 2004, <http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/>

[Sha02] R. P. Shankar and F. Armando 2002. "SWORD: A Developer Toolkit for Web Service Composition". In *Proc. of the Eleventh International World Wide Web Conference*.

[Sch05] Schmit, B.A., Dustdar, S. (2005). Towards Transactional Web Services. 1st IEEE International Workshop on Service-oriented Solutions for Cooperative Organizations (SoS4CO '05), co-located with the 7th International IEEE Conference on E-Commerce Technology (CEC 2005), 19 July 2005, Munich, Germany.

[Sta03] Steffen Staab, Wil van der Aalst, V. Richard Benjamins, Amit Sheth, John A. Miller, Christoph Bussler, Alexander Maedche, Dieter Fensel, Dennis Gannon, "Web Services: Been There, Done That?," *IEEE Intelligent Systems*, Jan/Feb 2003, *IEEE Computer Society Press*, pp. 72-85.

[Sch04] Christoph Schuler, Rogr Weber, Heiko Schuldt, Hans-Jörg Schek, "Scalable Peer-to-Peer Process Management - The OSIRIS Approach." *ICWS 2004*, 26-34

[Sch02] R. Schmidt. "Web services based execution of business rules." *Proc. of the Intl. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[Sch05] B.A. Schmit, S. Dustdar, (2005). "Towards Transactional Web Services," *1st IEEE International Workshop on Service-oriented Solutions for Cooperative Organizations (SoS4CO '05)*, co-located with the 7th International IEEE Conference on E-Commerce Technology (CEC 2005), 19 July 2005, Munich, Germany.

[Sin04] Elias Sinderson, Vish Magapu, and Ronald Mak, "Portal of NASAs Mars Exploration Rovers Mission: Middleware and Web Services for the Collaboratiive Information," Invited paper, In *Proc. ACM/IFIP/USENIX 5th International Middleware Conference*, Toronto, Ontario, Canada, October 18th - 22, 2004.

[Sha] Vijayalakshmi Shanmugam, Applications in Bioinformatics: Integration of Web Services for Microarray. Analysis.

[Sha04] Sohrab P Shah, David YM He, Jessica N Sawkins, Jeffrey C Druce, Gerald Quon, Drew Lett, Grace XY Zheng, Tao Xu, BF Francis Ouellette. Pegasys: software for executing and integrating analyses of biological sequences. *BMC Bioinformatics* 2004.

[Sch02] R. Schmidt. "Web services based execution of business rules." *In Proc. of the Intl. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[Ste03] Steele, R. A "Web Services-based System for Ad-hoc Mobile Application Integration," In Proc. of IEEE Intl. Conf. on Information Technology: Coding and Computing '03, 2003.

[Ste03] Steele, R. A, "Web Services-based System for Ad-hoc Mobile Application Integration," In Proc. of IEEE Intl. Conf. on Information Technology: Coding and Computing '03, 2003.

[Sha01] Shalom Tsur, "Are Web services the next revolution in E-commerce?", *Panel at 27th Very Large Scale Database (VLDB) conf.*, Roma, Italy 2001..

[Tal02] D. Talia, "The Open Grid Services Architecture: where the grid meets the WebT"" *Internet Computing, IEEE* , Volume: 6 Issue: 6 , Nov.-Dec. 2002 Page(s): 67 – 71

[Tsu01] halom Tsur, Serge Abiteboul, Rakesh Agrawal, Umeshwar Dayal, Johannes Klein, Gerhard Weikum, "Are Web Services the Next Revolution in e-Commerce?" (Panel), Proceedings of the VLDB Conference, Roma Italy, September 2001, Morgan Kaufmann Publishers Inc, 614 – 617.

[Tai04] Stefan Tai, Rania Khalaf, and Thomas Mikalsen, "Composition of Coordinated Web Services," In Proceedings of the ACM/IFIP/USENIX International Conference on Distributed Systems Platforms (Middleware 2004), Toronto, Canada, October 2004

[Tha01] S. Thatte, "XLANG: Web Services for Business Process Design," <http://www.gotdotnet.com/team/xml/wsspecs/xlang-c/default.htm>, May 2001.

[Udi02] Z. M Udin. and M. K. Khan, "A Framework for Collaborative Supply Chain: Level 1- Planning for Redesign". *Proc. of the Seventh Intl. Conf. on CSCCW in Design*. Rio de Janeiro, Brazil, 2002. p 325.

[Ver05] Kunal Verma, Kaarthik Sivashan-mugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Journal of Information Technology and Management," Kluwer Academic Publishers, Special Issue on Universal Global Integration, Vol. 6, No. 1 (2005) pp. 17-39.

[Ver05] METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services, Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth et.al J. Information Technology and Management, Special Issue on Universal Global Integration, Vol. 6, No. 1, 2005 pp. 17-39.

[Var05]Indrakanti, S., Varadharajan, V. & Hitchens, M. “Authorization Service for Web Services and its Application in a Health Care Domain,” International Journal of Web Services Research, Vol. 2, Issue 4, September 2005, pp. 94 – 119

[WFMC] Workflow Management Coalition, <http://www.wfmc.org>

[WSC02] “Web Services Coordination” (WS-Coordination 1.0), 2002, <http://www.106.ibm.com/developerworks/library/ws-coor/>

[WSCL] “Web Service Conversation Language” (WSCL) , HP Submission to W3C, <http://www.w3c.org>, 2002.T

[Wil00] S. Müller-Wilken, F. Wienberg, W. Lamersdorf , “ On Integrating Mobile Devices into a Workflow Management Scenario,” *In Proc. 11Pth Intl. Workshop on Database and Expert Systems Applications (DEXA'00), September 06 - 08, 2000 ,Greenwich, London, U.K., IEEE Computer Society Press*, pp. 186-192.

[Woh02] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede, “ Pattern based analysis of BPEL4WS”, *Technical Report FIT-TR-2002-04*, QUT, Queensland University of Technology, 2002.

[Wee05] Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, TonyStorey, et al.”Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More,” Prentice Hall, Paperback, Published March 2005, 416 pages, ISBN 0131488740.

[Woh03] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede “Analysis of Web Services Composition Languages: The Case of BPEL4WS,” *Lecture Notes in Computer Science, Conceptual Modeling- ER 2003*, pp. 200-215

[WorkflowPatterns] Workflow Patterns, Standard Evaluation <http://tmitwww.tm.tue.nl/research/patterns/standards.htm>, December, 2003.

[WSDL] *Web Services Description Language (WSDL)*. W3C Working Draft 10 May 2005, <http://www.w3.org/TR/2005/WD-wsdl20-20050510/>

[WST] “Web Services Transaction” (WS-Transaction 1.0), August, 2002, <http://www-106.ibm.com/developerworks/library/ws-transpec/>.

[WSC] “Web Services Coordination” (WS-Coordination 1.0), 2002, <http://www.106.ibm.com/developerworks/library/ws-coor/>

[WSCL] “Web Service Conversation Language” (WSCL) , HP Submission to W3C, <http://www.w3c.org>, 2002.

[WSCl] W3C “Web Service Choreography Interface (WSCl) 1.0,” <http://www.w3.org/TR/wsci/>, November 2002.

[WSFL] IBM Corporation, “Web Services Flow Language (WSFL1.0)”, <http://www-4.ibm.com/software/solutions/WebServices/pdf/WSFL.pdf>, May 2001.

[WST] “Web Services Transaction” (WS-Transaction 1.0), August, 2002, <http://www-106.ibm.com/developerworks/library/ws-transpec/>.

[Wan05] *Feng Wan; Munindar Singh*, Enabling Persistent Web Services via Commitments Information Technology and Management, January 2005, vol. 6, no. 1, pp. 41-60(20)

[You95] F. Yousfi, Bricon-Souf, N., Beuscart, R., Geib, J.M. “PLACO: a cooperative architecture for solving coordination problem in health care,” *In Proc. IEEE 17th Annual Conf. Engineering in Medicine and Biology Society, Sep 20-25, 1995*, Vol: 1 pp. 747 – 748.

[Yun98] Y.Yuan, Joseph B. Rose, and Norm Archer, “A Web-Based Negotiation Support System,” *Electronic Markets*, V. 8, N. 3, 1998.

[Yun98] Y.Yuan, Joseph B. Rose, Norm Archer, “A Web-Based Negotiation Support System”, in: Schmid, Beat F.; Selz, Dorian; Sing, Regine: EM - Electronic Contracting. EM – Electronic Markets, V. 8, N. 3, 1998.

[Zla03] Z. Zlatev, and P. Eck, “An Investigation of the Negotiation Domain for Electronic Commerce Information Systems,” *In Proc. of the 5th Intl. Conf. on Enterprise Information Systems, ICEIS 2003, Angers, France, April 2003*.