Georgia State University
Digital Archive @ GSU

Computer Science Theses | Department of Computer Science

1-12-2006

A System for Rapid Configuration of Distributed Workflows over Web Services and their Handheld-Based Coordination

Jaimini Joshi

Follow this and additional works at: http://digitalarchive.gsu.edu/cs_theses

Recommended Citation
Joshi, Jaimini, "A System for Rapid Configuration of Distributed Workflows over Web Services and their Handheld-Based Coordination" (2006). *Computer Science Theses.* Paper 18.

This Thesis is brought to you for free and open access by the Department of Computer Science at Digital Archive @ GSU. It has been accepted for inclusion in Computer Science Theses by an authorized administrator of Digital Archive @ GSU. For more information, please contact digitalarchive@gsu.edu.

A SYSTEM FOR RAPID CONFIGURATION OF DISTRIBUTED WORKFLOWS OVER WEB SERVICES AND THEIR HANDHELD-BASED COORDINATION

by

JAIMINI JOSHI

Under the Direction of Sushil K. Prasad

ABSTRACT

Web services technology has lately stirred tremendous interest in industry as well as the academia. Web services are self-contained, platform independent functionality which is available over the internet. Web services are defined, discovered & accessed using a standard protocols like WSDL, UDDI & SOAP. With the advent of Service-Oriented Architecture and need for more complex application, it became eminent to have a way in which these independent entities could collaborate in a coherent manner to provide a high level functionality. But the problem of service composition is not an easy one. One reason being the self-contained and loosely coupled interaction style, which happens to be the single most important reason for its popularity. We are proposing a prototype system for distributed coordination of web services. This system is based on the Web Bonds model for coordination.

The system, dubbed BondFlow system, allows configuration and execution of workflows configured over web services. Presently BondFlow system allows both centralized as well as distributed coordination of workflows over handhelds, which we claim as an engineering feet and is currently a unique work in this area.

Index words: Web services, coordination, composition of web services, BondFlow, Web Bonds, distributed systems

A SYSTEM FOR RAPID CONFIGURATION OF DISTRIBUTED WORKFLOWS OVER WEB SERVICES AND THEIR HANDHELD-BASED COORDINATION

by

JAIMINI JOSHI

A Thesis presented in partial fulfillment of requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2005

Copyright by
Jaimini L. Joshi
Master of Science (Computer Science)
2005

A SYSTEM FOR RAPID CONFIGURATION OF DISTRIBUTED WORKFLOWS OVER WEB SERVICES AND THEIR HANDHELD-BASED COORDINATION

by

JAIMINI JOSHI

Major Professor: Sushil K. Prasad
Committee: Alex Zelikovsky
Raj Sunderraman

Electronic Version Approved:

Office of Graduate Studies
College of Arts and Sciences
Georgia State University
December 2005

Dedicated to everyone who was part of this
For all the support

Acknowledgements

I would like to thank my advisor, Dr. Sushil K .Prasad, for his expert guidance. He was always receptive to new ideas and ready to explore options. He always helped me see the bigger picture thus helping me focus on the main issues in the system.

A special thanks to Janaka Balasooriya. He was a constant source of information and advice during the research.

Dr. Raj Suderraman & Dr. Alex Zelikovsky were kind enough to review the manuscript and provide me with fine pointers to meet the standards.

Table of contents

Acknowledgements ..... **v**

List of Tables ..... **viii**

List of Figures ..... **ix**

1 INTRODUCTION **1**

1.1 Motivation 1

1.2 Specific contributions 2

1.3 Organization of thesis 3

2 WEB SERVICES & THE PROBLEM OF COMPOSITION **4**

2.1 SOAP 4

2.2 WSDL 6

2.3 UDDI 7

2.4 Problem of Service Composition 8

2.5 Current Standards & their limitations 8

3 BACKGROUND **11**

3.1 Web coordination bonds 11

3.2 SyD Middleware Framework 12

4 BONDFLOW SYSTEM: ARCHITECTURE & DESIGN **14**

4.1 BondFlow Configuration System 15

4.2 BondFlow Runtime 22

5 BENCHMARK WORKFLOW PATTERNS & USER DEFINED CONSTRUCTS **31**

5.1 Control Flow Patterns 31

5.2 Extended Coordination Manager 35

6 FLAVORS OF BONDFLOW SYSTEM 40

6.1 Centralized PC based 40

6.2 Centralized iPAQ based 40

6.3 Distributed iPAQ based 41

7 SYSTEM EVALUATION 43

8 CONCLUSION & FUTURE WORK 48

9 BIBLIOGRAPHY 49

List of Tables

Table 1. WSDL Definition sections *7*

Table 2. WSCP Generation Scenario *16*

Table 3. Workflow Creation Scenario *18*

Table 4. MarkAndLock Messages *27*

Table 5. EnforceSubscriptionBond Messages *29*

Table 6. Configuring Extended Coordination Manager *37*

Table 7. Performance Measurements *47*

List of Figures

Figure 1. SOAP components interaction *6*

Figure 2. BondFlow Configuration System Modules *16*

Figure 3. BondFlow Runtime System Architecture..... *22*

Figure 4. Coordination Manager Interactions *24*

Figure 5. State/Instance Handler working..... *25*

Figure 6. Partial flow of activities for enforcing Pre-Negotiation Bond *26*

Figure 7. ECM deployment flow of activities *38*

Figure 9. Online Book purchase workflow..... *44*

Figure 10. Puchase Order Workflow[BPEL spec] *45*

Figure 11. Purchase order workflow modeled using Web Bonds *46*

1 INTRODUCTION

Web service choreography has spurred a lot of interest and attracting researchers from industry and university equally. It stands proven that web services can prove to be a good building block for complex application and allows high-level of customization. BondFlow system provides a rapid configuration and deployment environment of workflow over web services. BondFlow system is based on the Web Bonds Model [26]. The system allows expert users with high level of flexibility and novice user have ease of configuration. It is a good blend of abstraction and low-level artifacts. The system currently can be deployed in variety of target environment including handhelds.

1.1 Motivation

There is a need for a platform which allows user to easily configure and deploy complex workflow over web services. Web services are often seen as independent self contained entities. Coordination among these entities would require defining a layer of middleware which provides the necessary booking keeping services. Further the need of the day calls for distributed coordination of web services as oppose to centralized coordination, for reason such as bottleneck due to centralized control, lack of fault tolerance. The industry is divided among the proponent for centralized and distributed coordination. These models are generally known as Web services orchestration and web services choreography respectively. The main of the system was to provide easy configuration and deployment capabilities. Most of the current systems either are too complex, requiring expert knowledge, or just too high level, thus cutting on the level of flexibility. BondFlow system tries to strike a balance between the two extremes.

1.2 Specific contributions

BondFlow system claims to be unique in terms of implementing web services coordination over web services. Previous work [28] related to developing a wrapper based framework which can further be developed to provide true distributed coordination among web services. Further the current work also includes a innovation design for handling control flow patterns in workflows.

The listed below are the few contributions made by the work:

1) Prototype implementation of the Web Bonds Model i.e. Bonds, Coordination Context, WSCP.
2) BondFlow Pattern APIs for creating known control flow patterns and also arbitrary patterns.
3) Framework for configuring and deploying benchmark control flow patterns.
4) BondFlow Runtime ported over iPAQs for centralized coordination of web services
5) Integration of BondFlow Runtime with SyD Middleware allowing truly distributed workflow execution over web services.
6) Currently, BondFlow system prototypes are implemented for all the following target environments
    a. Centralized coordination over PC
    b. Centralized coordination over iPAQs
    c. Distributed coordination over iPAQs
7) A comparative view of the system with respect to BPEL.

1.3 Organization of thesis

In the next section, we talk about the various technologies involved and the current solutions to the problem of service composition. Section 3, discusses the architecture of the system in detail. Section 6 is devoted to the framework of implementing workflow patterns. We end the thesis, by providing system performance measurement in section 6 and conclusion in section 7.

2 WEB SERVICES & THE PROBLEM OF COMPOSITION

In its simple form a web service would mean service available over the internet. A functionality provided by a service provider which can be accessed in a standard and a platform independent way.

From this it has evolved in a fundamental building block used to create distributed applications. Lately, a new paradigm of software engineering has picked up interest; this is Software oriented architecture. As per this model each component in the system is modeled as a service which is self-contained and can be accessed using a set of standard protocols. Web services fit perfectly in this definition. Thus, allowing Web Services to become the platform for application integration. Applications are constructed using multiple Web Services from various sources that work together regardless of where they reside or how they were implemented. Web services mainly constitute of a set of standard protocols.

The main components of the web services model are the various protocols it supports. SOAP, WSDL and UDDI are the few commonly used ones. These protocols have been around since long now but have found it use more appropriate in web services stack. We take some time off to understand how these protocols work and how they come together in the web services architecture

2.1 SOAP

SOAP stands for Simple Object Access Protocol [19]. It is an XML-based messaging protocol which is used to access objects and services over the internet. The fact that XML is used as the method of communication makes it platform independent. Moreover, XML

is widely used as standard for exchange of information. SOAP was design with the following goals in mind [19]:

1) Simple one-way communication of messages packaged in XML
2) Use this paradigm to provide for RPC-type interactions, thus allowing synchronous request-reply mode of communication
3) Provide for allowing SOAP messages to the transported using the current transport protocols like HTTP, SMTP, etc.

The basic unit of exchange in SOAP is a SOAP message. A SOAP message consists of an optional header part and a required body part. A SOAP header contains various flags which may be used by intermediate node during the routing of the message. SOAP allows full freedom as to the contents of the SOAP body. Generally the contents of the body of the message are governed by the SOAP router and the application sending the message. In a typical web service call the SOAP message contains the name of the operation along with the serialized parameters for the operation.

SOAP allows two types of interaction styles, RPC-style and Document-style. RPC-style is used when an application wants to have synchronous operation call over the internet. The message body, in this case, would contain the method name and the parameter values for the operation. The other type of interaction style is Document-style. Document-style message exchanges have a fixed format for defining the data being exchanged and this is generally modeled as a document e.g. purchase order, invoice receipt, etc. The format of the document is predefined and agreed upon by the interacting parties. The actual way the data is formatted in XML is defined by the *encoding rules*

used in the SOAP message. SOAP supports two types of encoding, *literal* and *SOAP encoding*. But applications have are allowed to use different encoding rules if the need be.

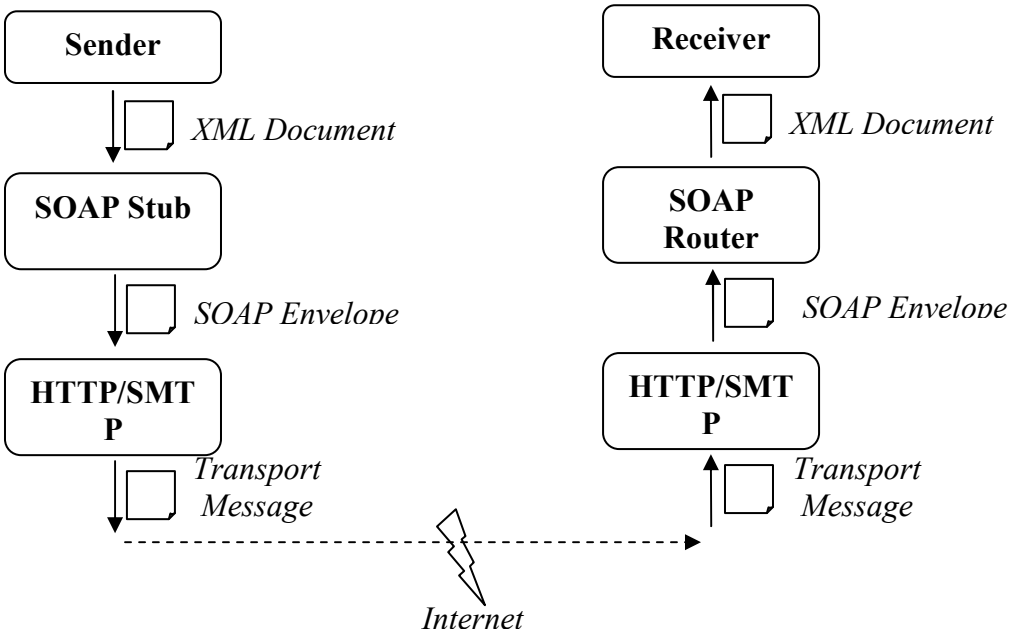

Figure 1. SOAP components interaction

2.2 WSDL

Web Service Definition Language. It is the *IDL* for web services. WSDL defines the operation a web service exposes in a platform independent way. The documents describing a web service is divided into part, abstract part and the concrete part. The abstract part defines the types, messages, operation and port types of the service. And the concrete part actually binds each of the port type defined in the abstract part to actual service ports i.e. to a fixed implementation of the service. Let us discuss each of the above mentioned part in detail.

Table 1. WSDL Definition sections

Abstract Part	**Type**	Simple & Complex data types defined using XML Schema
	Messages	Message definition and their format
	Operation	Operation definition(name and input-output messages)
	Port Types	Logical grouping of operation into port types
Concrete Part	**Bindings**	Provide binding information to actual communication protocol (e.g. SOAP), transport protocol, encoding style and interaction style
	Ports	Provide actual network address for each port type
	Service	Logical grouping of ports

2.3 UDDI

Universal Description Discovery & Integration service is a discovery service for web services. Web services when registered with the directory service are available for search and use by end users. UDDI stores service information including service metadata which can facilitate criterion based service search. UDDI defines set of APIs for publishing and querying the registry, also provides with a set of data structures which are involved in these operations. It should be noted that UDDI registry is also exposed as a web service and the WSDL of which is easily available.

2.4 Problem of Service Composition

Web service composition means grouping of related basic services to provide high level functionality. With the advent of internet and web services, we now have a plethora of services which provide different type of services like payment service, stock quote service, air ticket booking service, etc. Now what if we could put these services together in some particular order and provide a new service which provides a high level functionality. E.g. we compose a new service air ticket booking and payment service. This new service uses an *air ticket booking service* which books a ticket and then the payment details are transferred to the payment settlement service. Once the payment is settled the user is informed about the flight details. This is a simple case of web service composition but it gives a good overview of how well we can model the various business processes of an organization. Service composition strives to provide the end user with high level of customization and state-of-the-art services to satisfy the requirements. The trend in service composition, now days, is to provide for automatic or on-the-fly composition. This are the cases when the services to be used are not known until unless the runtime. The service instead of being statically bound are *discovered* at runtime and executed then. Such discovery of services is based on various utility measuring criterions such as QoS, cost, effectiveness to the problem at hand and other.

2.5 Current Standards & their limitations

Service composition, especially web service composition, has attracted industry attention since late. This is partly because of the sudden increase in deployed web services available for use to the end user. We have seen a lot of attempts by different groups to come up with a good solution to the problem of service composition. Each one

claiming to be better than the other but essentially saying the same thing with different set of words. Currently, we have some widely used service composition languages like Business Process Execution Language (BPEL), Yet Another Workflow Language (YAWL), Web Service Choreography. Apart from this there are certain standards which are still in being formalized like Web service coordination framework (WS-CF), Web Service – Choreography Description Language (WS-CDL). Most of the standards overlap in the domain of problem covered. Of all these, BPEL is de facto for web service composition. BPEL specification defines an XML-based language for service composition. BPEL working can easily be understood by comparing it with the traditional structured programming model. A basic unit of execution in BPEL is *BPEL Process*. A BPEL process is defined with the help of an XML formatted file using the constructs provided by the BPEL specifications. This specification has all the basic constructs which a programming language has like loops, conditional block execution, switch statements, etc. A BPEL process is allowed to make calls to other BPEL processes and other independent web services. BPEL process itself is modeled a web service, thus associated with a BPEL process we have a WSDL file which defines abstract part of process. The actual binding information is available only at deploy time. A BPEL engine provides the runtime and deployment environment for a BPEL process. It is the BPEL engine which provides the BPEL process with the actual binding information. One of the main reasons for BPEL's success is because it is a convergence of two different standards provided by two big software giants, XLANG from Microsoft and Web Service Flow Language from IBM. And further its use of widely used standards like XML Schema, WSDL also attribute to the success. But what BPEL tries achieve is what we call Web

service orchestration. Thus we have a problem of centralized controller, which in case of BPEL is the BPEL process. YAWL is also one graphical workflow language with support for web services. YAWL does not use XML for process definition rather uses a proprietary method of object persistence. YAWL was designed based on an extensive research on control and data flow patterns. The research group has proposed an exhaustive set of control flow patterns [29]. YAWL, thus, has special constructs for each of this pattern and thus is restricted by it. The same is with BPEL specification. Most of the composition languages tend to restrict the user to the constructs available in the language and provide less of flexibility in terms of scalability. BondFlow system tries to overcome this by providing an easy to use plug-in based architecture [section 5] to support not only the known control flow patterns but also provide for arbitrary patterns in the workflow.

3 BACKGROUND

Here, we briefly discuss web coordination bonds and the SyD middleware framework.

3.1 Web coordination bonds

Web coordination bonds [4, 26] are a set of primitives for web service coordination/choreography. 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. While it is convenient to think of an entity as a row, a column, a table, or a set of tables in a data-store, the concept transcends these to any object or software component, and here we specifically consider web services. There are two types of web bonds: subscription bonds and negotiation bonds. Subscription bonds allow automatic flow of information and control 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 or event flows.

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 specified set of parameters); "Mark *X*" refers to an attempted change, which triggers any associated bond without an actual change on *X*.

Subscription Bond: Mark *A*; If successful, Change *A* then Try: Change *B*, Change *C*. A ``try'' may not succeed.

Negotiation-and Bond: Change *A* only if *B* and *C* can be successfully changed. (Implements atomic transaction with "and" logic).

Similar semantics can be defined for Negotiation-or and “xor” bonds. Likewise, these logical primitives are available for subscription bonds. Additionally, user defined rules and evaluating conditions can be incorporated. The “and” logic is the default constraint, if unspecified. Formal treatment of Web bonds and their firing rules are discussed in [26].

We have established that web bonds have the modeling power of extended Petri nets and they can express all the benchmark patterns for workflow and for inter-process communication; a feat that almost all previously proposed artifacts and languages are not capable of comprehensively, thus proving that web bonds are superior despite their simplicity [26].

3.2 SyD Middleware Framework

System on Mobile Devices (SyD) middleware [1] is a new platform technology that addresses the key problems of heterogeneity of device, data format and network, and of mobility. SyD combines ease of application development, mobility of code, application, data and users, independence from network and geographical location, and the scalability required of large enterprise applications concurrently with the small footprint required by handheld devices. SyD separates device management from management of groups of users and/or data stores. Each device is managed by a SyD deviceware that encapsulates it to present a uniform and persistent object view of the device data and methods. Groups of SyD devices are managed by the SyD groupware that brokers all inter-device activities, and presents a uniform world-view to the SyD application. The SyD groupware directory service enables SyD applications to dynamically form groups of objects hosted

by devices. The SyD deviceware enables group communication and using SyD listener. The footprint of the entire SyD kernel code is 112 KB, out of which only 76 KB is currently device-resident; the rest is for directory and global event handling. For even smaller devices, this can be further reduced to 42 KB. The execution time workspace used by SyD is 4-8 MB, exclusive of JVM and OS. Web bonds have been employed to model and enforce work workflow control flow/data and other dependencies in our BondFlow system whilst SyD middleware has been employed as a workflow deployment and execution platform for handheld devices.

4 BONDFLOW SYSTEM: ARCHITECTURE & DESIGN

BondFlow system is a realization of Web Coordination Management Architecture proposed in [26] using Web Bonds. BondFlow system proposes a coordination management system over web services, services in general sense, for implementing distributed workflows. We propose that by abstracting the user from various requirements necessary for distributed coordination of entities, we can dramatically reduce the programming exercise involved in developing distributed systems over the internet. BondFlow system allows novice as well as expert users to configure and deploy workflows over web services. We claim that the basic artifacts defined by the web bonds model are abstract enough to provide both ease of use & flexibility for modeling complex workflows. Furthermore, the design of the system makes a clear separation between the configuration and executions of the workflows. Thus, allowing us to provide support for various runtime environments without changing the configured workflows. At present we have three different target environment of the runtime system namely, centralized coordination over PCs, centralized coordination over iPAQs and distributed coordination over iPAQs.

At the heart of our system is a *Web Service Coordination Proxy.* WSCP is a software abstraction over a web service. WSCP object represents the actual web service in the workflow. The proxy has the intelligence of managing the bonds dynamically and enforcing dependencies to cater to the needs of a particular workflow. All invocations on a web service have to pass through the BondFlow system generated WSCP. It is this indirection that allows us to bring transparency to the system and hide the necessary

coordination and communication logic behind it. WSCP itself provides the same interface what the web service provides to the outer world. These WSCPs are generated for each of the services which are added to the system. The process of WSCP generation takes negligible time and has small footprints [28], small enough to be ported on to handhelds.

BondFlow Architecture Overview:

The BondFlow system is a platform which allows distributed execution and coordination of workflows over web services. As described before it works on the principle of indirection by creating a software abstraction over web services which allows us to embed the coordination logic between actual web service calls. The core of the system consists of a runtime execution environment, *BondFlow Runtime,* for executing these software WSCPs. One other sub-system of BondFlow system is the BondFlow Configuration System (BCS) which handles the workflow creation and deployment. BCS is the interface of the BondFlow system to the user.

4.1 BondFlow Configuration System

BCS is the point of interaction between the end user and the system. BCS allows creation of new workflows and deploying them over-the-network (optional). The BCS takes care of the following tasks: Web Service Coordination Proxy Object (WSCP) generation, creation of workflows, deploying workflows and editing existing workflows. We explain each of these tasks in detail.

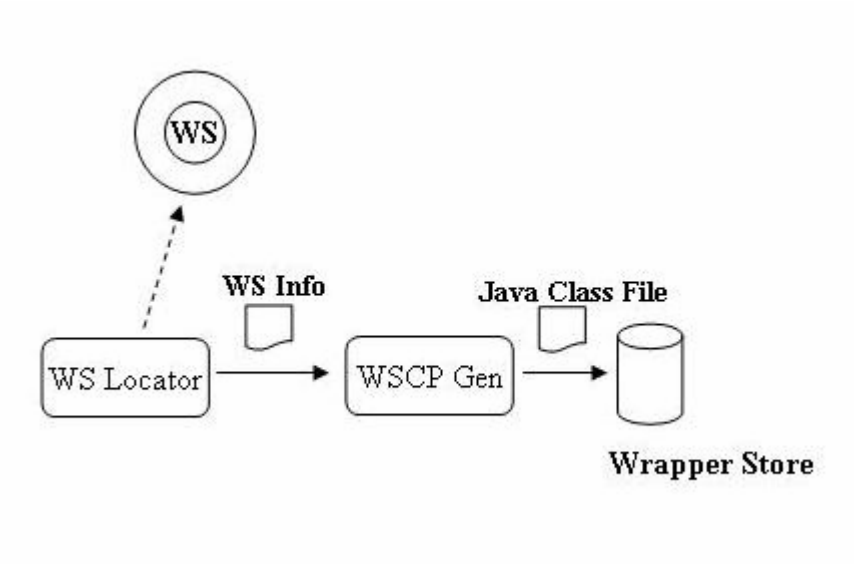

Figure 2. WSCP Generation Scenario

WSCP Generation Task:

This scenario depicts the process when a new service is added to the system in order to be part of some collaborative application. Collaboration over a set of services is only possible if they have a corresponding WSCP associated with them. Thus, it is a necessary registration procedure a web service needs to pass through so as to be available for use within the system.

Table 2. WSCP Generation Task

Modules : WS Locator, WSCP Generator *Input :* WSDL URL *Output :* WSCP Java File & Class File, Service XML file *Process:* 1. WS Locator module uses the WSDL URL to obtain the WSDL file.

2. Once the WSDL file is available, the WSCP Generator parses this file to obtain the service specific information e.g. Operations names, parameters, return types. The extracted information is then stored in the XML repository for the service. The parsing of WSDL is accomplished using the WSDL4J APIs.
3. Using the parsed information and a pre-defined template a WSCP class is created. The web service operations have one to one correspondence with the method of the WSCP class. The name of the WSCP class is same as that of the web service it encapsulates.

Format of Service XML file:

```
<WebService>
 <WSDef>
  <WsdlUrl>http://www.xmethods.net/sd/2001/
     CurrencyExchangeService.wsdl</WsdlUrl>
  <WSName>CurrencyExchangeService</WSName>
<LocationURI>http://services.xmethods.net:80/soap</LocationURI>
  <NamespaceURI>urn:xmethods-CurrencyExchange</NamespaceURI>
  <Method mid="1">
  <MethodName>getRate</MethodName>
   <ParamType>string country1</ParamType>
   <ParamType>string country2</ParamType>
   <ReturnType>float Result</ReturnType>
  </Method>
 </WSDef>
```

```
</WebService>
```

The WSCP generation process depends on the target deployment configuration.

In BPEL:

BPEL also defines similar construct for allowing the necessary abstraction to web services by defining a BPEL process. Each BPEL process is a special web service defined using WSDL but it only contains the definition for port types and no binding information. Actual binding information is added later by the BPEL engine, which further provides implementation of each port type as defined by the workflow specification file.

Workflow Creation Task:

With the help of Subscription and Negotiation bonds, we can model the control and data dependencies among the web services. More details on web bonds model is available in our previous work in [4]. Workflow configuration process starts by creating bonds among methods of selected web services through the proxy object to reflect dependencies. Bonded web services represent a configured workflow with substantial coordination logic embedded. In BondFlow architecture, each proxy object has its corresponding "Bond" repository to reflect its own dependencies. Bond constraints are specified while creating and necessary information is stored in persistent XML files.

Table 3. Workflow Creation Task

Modules : Bond Creation
Input : Bond Type and other parameters
Output : Generate `CoordinationContext` for each service

Process:

1. User is prompted to put in the name of the workflow.
2. User is than is asked for a pair of web service names & respective methods which would act as source and destination methods for a bond and various other parameters as type of bond, trigger, and boolean logic.
3. All this information become part of the coordination context of the source web service as its coordination manager is responsible to enforce these bonds. The coordination context of a service is stored in the XML repository for that particular service.
4. Step 2-3 goes until whole of the workflow is not fully modeled.

Format of Service XML file containing the coordination context information:

```
<WebService>
 <WSDef>...</WSDef>
 <CoordinationContext>
   <Application name="TestApp">
    <Bond bid="1" status="true">
      <SrcMethod>getRate</SrcMethod>
      <DestMethods>
       <DestMethod>StockQuoteService.getQuote</DestMethod>
      </DestMethods>
      <Type>S</Type>
      <Boolean>AND<Boolean>
      <Trigger>Y</Trigger>
```

```
      </Bond>
     </Application>
   </CoordinationContext>
</WebService>
```

In BPEL: BPEL is a XML based definition language. In order to define a BPEL process, a user needs to create a BPEL file which contains the flow specification in XML format. BPEL models the control flow between activities using *links*. Service link types are special types of links which allows the user to assign specific roles to a service which it may assume during an interaction. User can define partners to a service using the service link types. It should be noted that actual binding of partner links to a service implementation is done at runtime and it is a feature which a BPEL implementation needs to support.

Deploying Workflows:

BondFlow system can be deployed under various different coordination and hardware configurations. To classify according to mode of communication, Wired & Wireless. Furthermore, we can have both centralized as well as distributed coordination WSCP deployment scenarios. It should be noted that the way the workflow is configured does not depend on the deployment configuration. Only the WSCP creation process is deployment configuration specific.

The BondFlow system boasts of distributed coordination among web services. This is achieved by distributing each of the generated WSCPs over the network. However, the WSCPs should also be accompanied by the service XML file as it contains

the coordination context for the service. We can understand the relation between this two as, WSCP knows how to enforce a bond and coordination context tells it when. The process of deploying the WSCP and the service xml file can be done offline e.g. one way of doing this would be hosting on a machine connected to the internet using FTP. It should be noted that the machine hosting the WSCP needs to have BondFlow Runtime environment installed. Further a WSCP must be registered with a pre-known directory service, in order to allow location dependents communication between WSCPs. As will be discussed further current implementation uses the directory service provide by the SyD middleware [1] for registering the WSCP. WSCP are modeled as SyD Application Objects (SyDAppO) which are then registered with SyDDirectory service. Communication between these SyDAppOs is facilitated by SyDListener and SyDEngine. The design of the system abstracts the protocol used between WSCPs. Given that, it is also possible use other messaging protocols like SOAP to implement Inter-WSCP communication. For centralized runtime system, the Inter-WSCP communication is modeled as in-memory calls of the process. More detail on different flavors of the system is given in section 6.

In BPEL: Once BPEL process is defined and all the dependencies collected. A BPEL engine can be used to deploy a BPEL process. BPEL Engine is generally available web application run on an application server. And the actual steps needed to deploy a BPEL process depends on the implementation of the engine. When a BPEL process is deployed all references to services needs to be resolved by providing the WSDL file of the actual service implementation. Due to the centralized nature of BPEL process, it becomes each to incorporate auditing and monitoring features in the system. These features are at the

discretion of the BPEL Engine vendor and no way part of the specification but prove as a handy tool when deploying long running processes spanning over multiple web services.

4.2 BondFlow Runtime

BondFlow runtime is a realization of the web bonds model as proposed in [4]. It enforces control and data dependencies modeled using web bonds on a web service. BondFlow WSCP uses the coordination context information stored in the service XML file and calls system routines on BR to enforce this coordination constraints. The runtime environment of the system essentially consists of the specific modules and the WSCP class interacting in a defined manner.

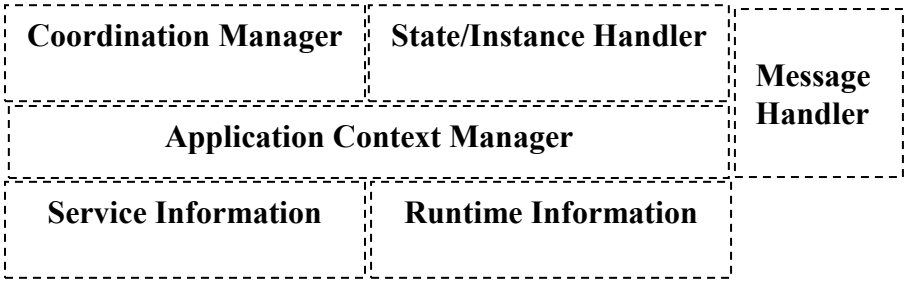

Figure 3. BondFlow Runtime System Architecture

Application Context Manager:

Responsibility: Application context manager allows us to define multiple workflows over a same web service WSCP. It is responsible to provide the WSCP class and all other modules of the system with an application specific view.

Software Implementation: WSCP class

BondFlow system allows different workflows to be deployed over a web service thus allowing more multiple applications sharing the BR on a machine. Application Context manager logically divides the BR for each of the applications running under the system. Context information for an application would include a unique identifier `appid`. Further, it also includes application specific parameters which may have been defined prior to deployment of workflow. Currently the application context manager is in the minimalist form and work is undergoing as to what all can be the part of this structure.

Coordination Manager:

Responsibility: Coordination manager is responsible for enforcing the web bonds defined on a service. For this, the module uses the coordination context information available from the service XML file.

Software Implementation: WSCP class, `Engine` *Package in BR class hierarchy, AXIS APIs, NanoXML Parser*

Each operation of a web service may have a set of bonds defined over it. This information along with other parameters, as defined in section 2, is stored in the service XML file. This information essentially forms the coordination context of a method. Coordination manager is responsible for enforcing these bonds over the method depending on the coordination context. Coordination manager looks up the bond information defined over the web service method for a particular application. Depending on the type of bond appropriate BR system call are made to enforce them. Coordination manager uses the message handler module to send out messages to BR of the destination web service for a particular bond. Further coordination manager also interacts with the

state and instance handler to update the state information of a method of the web service of the WSCP.

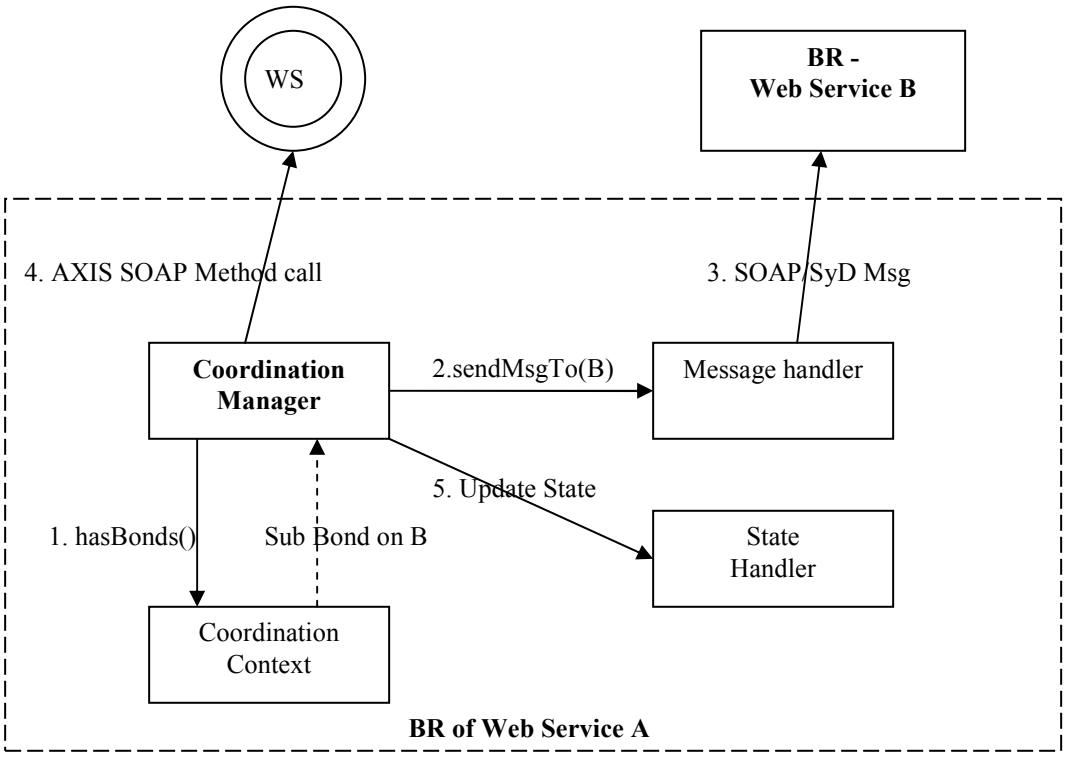

Figure 4. Coordination Manager Interactions

State/Instance Handler:

Responsibilities: State/Instance handler is responsible for maintaining state information for operations of web service and also the instance related information for a web service. This information is needed to provide for certain patterns in workflows.

Software Implementation: WSCP classes, Service runtime XML, NanoXML.

BondFlow system needs to maintain runtime state information of each of the methods. The state information for a method consists of execution status, method return

values under the current run, lock status of method, etc. This state information relate to a particular run of the workflow. The state information helps the coordination manager enforce bonds on particular method of a web service. State/Instance handler stores all the state information for the web service as a XML file under `/runtime` directory with the name of the service.

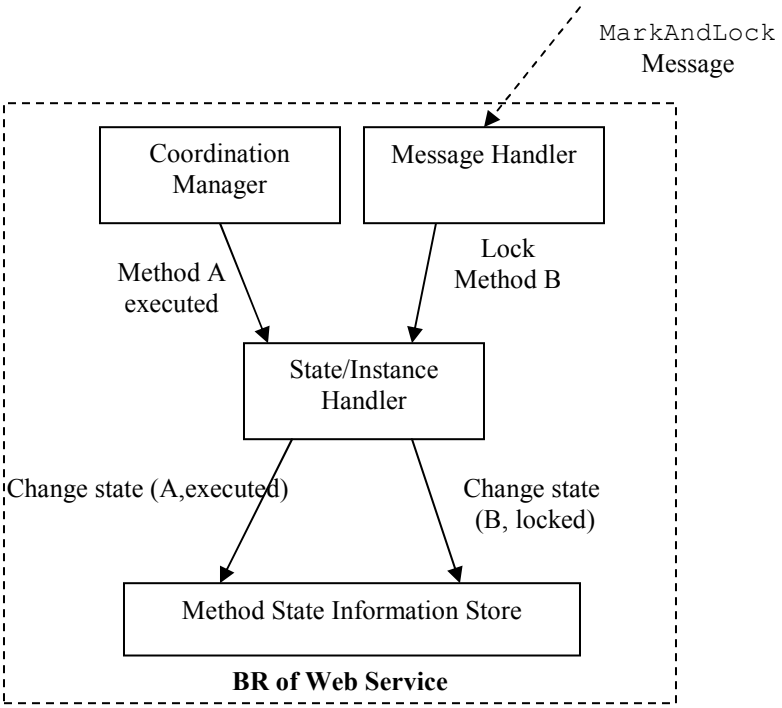

Figure 5. State/Instance Handler working

The `<wsname>.xml` files for the current scenario shown in Listing 1.

```
<apps>
 <app name="TestApp">
  <getQuote hasExecuted="true" return=""/>
  <someOther status="locked"/>
 </app>
</apps>
```

Listing 1. /runtime/<wsname>.xml file for <WSNAME> web service

Message Handler:

Responsibilities: Message handler is essentially a communication handler for Inter-WSCP communication. Message Handler is necessary for distributed deployment of WSCPs. WSCP could implement any of the messaging protocols including SOAP or SyDDoc.

Software Implementation: XML Message Parser

WSCPs in the BondFlow system communicate by exchanging messages over the internet. WSCPs use message as a means of passing of control and data. Moreover, it also has functionality similar to that of an asynchronous RMI handler as it also allows invoking operation on other WSCPs e.g. WSCP A may send a message to WSCP B asking it to mark and lock a method. This can be done by sending a `MarkAndLock` type message to the message handler of B. These messages are formatted as SOAP messages. BondFlow system defines various types of messages and formats for each of them. Due to space considerations, we will illustrate only a few message types here.

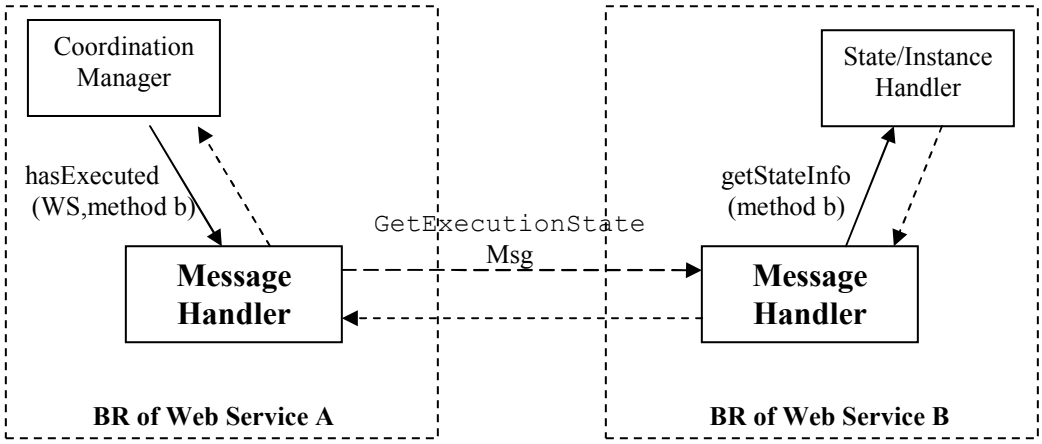

Figure 6. Partial flow of activities for enforcing Pre-Negotiation Bond

It should be noted we can have custom messaging solution implemented instead of SOAP and this is possible with out disrupting the working of other components of the system. Moreover, if the base platform has support for inter-application communication, BondFlow system can be made to adapt to use that too. This is would become evident in the later section on SyD.

Sample BondFlow SOAP Message Formats:

Here we give examples of how a SOAP messages generated by SOAP based message handler. (Currently our system does not have implementation of a SOAP message handler)

Table 4. MarkAndLock Messages

MarkAndLockRequest

`<env:Envelope xmlns:env=http://www.w3.org/2002/06/soap-envelope>`
`  <env:Header>`
`  <applicationContext`
`  env:role="http://www.w3.org/2002/06/role/ultimateReceiver"`
`  mustUnderstand="true">`
`  <appid>TestApp</appid>`
`  <source>`
`  <service>TemperatureService<service>`
`  <method>getTemp<method>`
`  </source>`

```
    </applicationContext>
    <messageType
      env:role="http://www.w3.org/2002/06/role/ultimateReceiver
      "
      mustUnderstand="true">
      MarkAndLockRequest
    <messageType>
  </env:Header>
  <env:Body>
    <MethodName>methodName</MethodName>
  <env:Body>
</env:Envelope>
```

MarkAndLockResponse

```
<env:Envelope xmlns:env=http://www.w3.org/2002/06/soap-
envelope>
  <env:Header>
    <applicationContext
      env:role="http://www.w3.org/2002/06/role/ultimateReceiver
      "
      mustUnderstand="true">
        <appid>TestApp</appid>
         <source>
           <service>CurrencyExchangeService<service>
           <method>methodName<method>
```

```
        </source>
    </applicationContext>
    <messageType
      env:role="http://www.w3.org/2002/06/role/ultimateReceiver
      "
      mustUnderstand="true">
      MarkAndLockResponse
    <messageType>
  </env:Header>
  <env:Body>
    <status>true<status>
  <env:Body>
</env:Envelope>
```

Table 5. EnforceSubscriptionBond Messages

```
<env:Envelope xmlns:env=http://www.w3.org/2002/06/soap-
envelope>
  <env:Header>
    <applicationContext
      env:role="http://www.w3.org/2002/06/role/ultimateReceiver
      "
      mustUnderstand="true">
        <appid>TestApp</appid>
        <source>
```

```
          <service>TemperatureService<service>
          <method>getTemp<method>
        </source>
    </applicationContext>
    <messageType
      env:role="http://www.w3.org/2002/06/role/ultimateReceiver
      "
      mustUnderstand="true">
      EnforceSubscriptionBond
    <messageType>
  </env:Header>
  <env:Body>
    <MethodName>methodName</MethodName>
  <env:Body>
</env:Envelope>
```

5 BENCHMARK WORKFLOW PATTERNS & USER DEFINED CONSTRUCTS

The theoretical underpinning of the BondFlow system is web coordination bonds. This section demonstrates modeling benchmark workflow control flow patterns using web coordination bonds. An attempt is made to provide a comparative view of modeling these constructs in Web Bonds and BPEL. We note that an existing workflow modeling framework called "YAML" 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.

In the paper [29], they have come up with a set of control patterns which are claimed to the exclusive set of constructs that would be needed any workflow. The patterns classified as per their nature are described below.

5.1 Control Flow Patterns

Basic Control Flow Patterns:

Basics control flow patterns capture basic split and join constructs. These constructs are relatively easy to implement and almost all the workflow models have mechanisms to support them. Here we briefly describe each pattern and then the

implementation of parallel split and simple merge construct have being presented in this section.

Sequence: An activity of a workflow is enabled after completion of another activity the same workflow.

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.

Synchronization: Synchronization is a point in a workflow where multiple control paths converge into a single control.

Exclusive Choice (XOR Split): XOR-Split is a point in a workflow where one of possible paths is selected.

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

Advanced Synchronization Patterns:

In advanced synchronization models, the problem arises as the split node can activate *m* out of *n* paths where $0 \leq m \leq n$. When it comes to the synchronization, synchronization node needs to know which paths to synchronize or whether synchronization is needed at all. Some cases, synchronization need to be done based on different merging criteria. Thus, synchronization is a significant issue in workflow modeling and has gained considerable attention. There are four advanced synchronization patterns.

Multi choice: A point in a workflow where one or several paths will be chosen based on some selection criteria.

Synchronous 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 merges 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

Multiple instance patters requires workflow activity to instantiate several instance of the activity and some cases instances need to be synchronized under various conditions before proceeding to the next activity of the workflow. There are four patterns involving multiple instances.

Multiple instances without synchronization: For any workflow activity, multiple instances of that activity can be created. These activities and independent and do not need to synchronize.

Multiple instances 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.

Multiple instances 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. Difficulty here is that numbers of instances is not known at the design time.

Multiple instances 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. It becomes more difficult due to the fact that numbers of instances is not known at the design time.

State Based Patterns

Characteristics of state based patterns. Description of three state based patterns.

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.

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 is executed in any order. Importantly, all the activities will be executed. Order is not know before runtime.

Structural Patterns:

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

Implicit terminator: 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

Web Bonds claim to be expressive enough to model all of the above patterns a feet only few workflow system are above to achieve. A detail discussion of the expressive power of web bonds is available in [26]. As mentioned earlier, workflow systems like YAWL provide explicit constructs to model and execute these patterns. This limits the system to the available implementation of these constructs. BondFlow system proposes a scalable design to provide support for the pattern which allows the a novice user to select from different implementations of the same construct depending on the requirement of the application and also always expert user to create new arbitrary patterns specific to an application. Under this scheme of things, our system allows to extend the default coordination context manager for certain scenarios. Thus, allowing a plug-in architecture which extends the scalability of the system.

5.2 Extended Coordination Manager

The Extended coordination manager defines 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. It should be noted here that the extended coordination manager is just a layer above the system coordination manager and thus is restricted by the services of the lower layer. Putting it

differently, it also will talk in terms of Subscription and Negotiation Bonds. The way these managers can be developed actually is not difficult or does not require heavy development exercise. The BondFlow system provides a common interface where new coordination manager can be plugged-in. This is achieved by providing 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 manager is defined a JAR file. This package contains the following:

1) roles.xml: This file contains definition of all the roles and their binding to specific manager classes. It also defines a set of properties which the role needs as input.
2) Set of class files: This class files relate to each role defined in `roles.xml`. There are no restrictions as to the name of the class files. The mapping between a class file and the respective role is done in metadata XML file. Each class must implement from `IPattern` interface, which defines the various callback methods for plugging in this manager.

The format of the metadata XML file is as follows:

```
<patterns>
 <pattern>
  <name> Discriminator </name>
  <desp> This manager allows deploy the discriminator pattern
</desp>
```

```
  <roles>
   <role>
    <name>Discriminator</name>
    <desp>The node acts as the discriminator node </desp>
    <classfile>NDisc.Disc</classfile>
    <execute>before<execute>
    <properties>
     <property name="n" type="java.lang.Integer">
    </properties>
   </role>
  </roles>
 </pattern>
</patterns>
```

Listing 3: Role definitions file for Discriminator role.

After preparing the JAR file it is stored in the `/plug-ins` directory of the BCS. The BCS could then be used to assign these roles to different web services. Further, the BCS further allows the user to configure the roles based the properties specified in the definition file. BCS is responsible for generating dependency.

Table 6. Configuring Extended Coordination Manager

Modules : , Plug-in handler, Main Module, Message handler, UI
Input : Extended Coordination Plug-in,
Output : Updating of Service XML file at the WSCP locations, Deployed dependencies at the WSCP locations
Process:

1. User using UI initiates the Plug-in Handler to use an Extended Coordination Manager (ECM).
2. Plug-in Handler brings up the UI for configuring the ECM. This configuration involves assigning roles to web services and providing values for extra parameters like properties for the role.
3. Plug-in handler than, for each web service which is assigned a role, creates a dependency store in which its stores the class files needed to execute that role and `role_runtime.xml`.
4. These dependencies are then deployed to individual WSCP locations and placed in `/runtime` folder of BR.

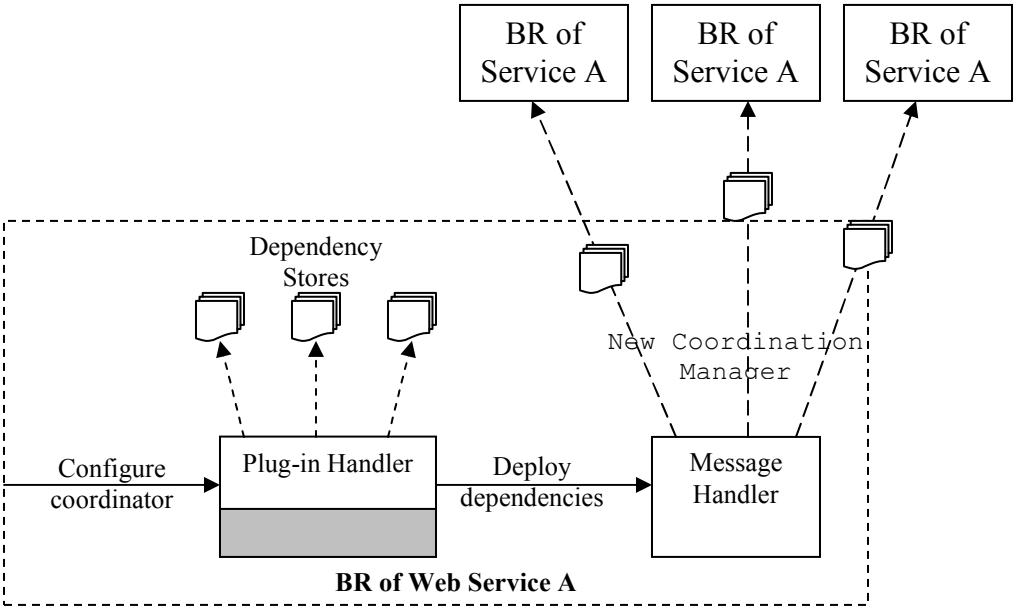

Figure 7. ECM deployment flow of activities

The concept of extended coordination manager is still in its experimental phase and needs more dwelling into but is quite promising. The design has allowed the BondFlow system to realize execution of very complex patterns like the Milestone, Discriminator and N of M pattern.

6 FLAVORS OF BONDFLOW SYSTEM

BondFlow system can be deployed in varyingly different configuration environments.

6.1 Centralized PC based

In this configuration, each wrapper is deployed on the same host and communication among wrappers is carried in form of in-memory calls. The centralized version, thus, does not have communication over head for Inter-wrapper communication. This setup allows the user to configure as well as deploy wrappers. Configuration subsystem of the BondFlow system (BCS) is only available on PC and thus cannot be used directly on handhelds.

Software Platform: JDK 1.4, Apache AXIS, NanoXML 2.21, WSDL4J

6.2 Centralized iPAQ based

The footprints of WSCP is very small [section 6] thus it is easy to port it to small devices. Following this, we ported the centralized PC based BondFlow system onto iPAQ. This required considerable amount of tweaking in the system so as to match the implementation as per the restrictions on a handheld. This included switching to a lightweight SOAP handler, ksoap 2. Moreover, the JVM available for iPAQ did not support JDK 1.4. Thus we had to practically redesign some of the components to fit them within the API set defined by JDK 1.2. As an end result we had a BondFlow System which allowed execution of workflows defined over web services on a handheld. Due to GUI restrictions and difficulty of use, BCS version for handhelds was not forayed.

Software Platform: JDK 1.2, ksoap2, NanoXML 2.21, Jeode JVM

6.3 Distributed iPAQ based

The distributed version of BondFlow system was realized by using SyD middleware [1] as the platform for distributed coordination. The BR used the services provided by the SyD middleware thus allowing it to register the WSCPs with the SyD Directory service and accessing them over-the-net in a location independent manner. For us to achieve this, we modeled the WSCP objects as SyD Application Objects (SyDAppO). SyDAppO are registered with the SyD Directory service, thus allowing all other WSCPs (SydAppOs) to access it using the SyDListener service.

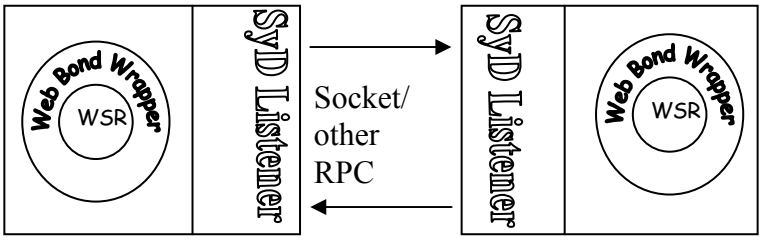

Figure 8. Wrapper Object resides in Java enabled Mobile Devices

SyD listener is a lightweight module in our SyD middleware framework for enabling mobile devices to host server objects and allow peer-to-peer communications among them. Different SyD application objects can communicate with each other through SyDListener. SyDDirectory allows registration of proxies and further lookup of this proxy objects. SyDDirectory maintains its own database to store information about all the SyD application objects and delivers location information of devices and services on the fly. It keeps track of application objects and their associated devices. SyD objects can lookup for remote objects through SyDDirectory. SyDEngine facilitates the object to actually invoke a remote objects. SyDListener keeps listening for any connection requests and delegates the control to SyDEngine module

WSCP Registration with SyDDirectory: This process is done with help of a helper class `ManageWrapper`. This class takes in the WSCP name and registers it using SyDRegistrar service. The SyDRegistrar creates a XML document, SyDDoc, which contains all the operation names and their parameters. This document is then sent to the SyDDirectory service.

Invocation of WSCP-over-SyD: Inter-WSCP communication is carried out by invoking the SyDEngine, which allows objects to make RPC-style calls in a location independent manner. An invocation call from a WSCP is first sent to SyDDirectory to lookup the location of the device which is hosting the required service, then the SyDEngine invokes the SyDListener on the server device and passes it a XML document, SyDDoc, which contains the operation invocation details. The SyDListener uses the local RMI registry to locate the object, invokes the required method and return the result to the client device. Due to the modular design of the BondFlow system it was easy to scale the design to a distributed environment. The message handler module was implemented to be SyD aware.

Software Platform: JDK 1.2, SyD Middleware, ksoap2, NanoXML 2.21, Jeode JVM

7 SYSTEM EVALUATION

The BondFlow system has been prototyped using Java 1.4 and the footprint of the BondFlow 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 for handled version).

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 XML parser for JAVA. Various publicly available web services including Xmethod's SOAP based web services (http://www.xmethods.net/) have been used for 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.

Case study workflows: We 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: For this workflow, we have used real web services available in xmethods.com and few other service directories. Here, "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.

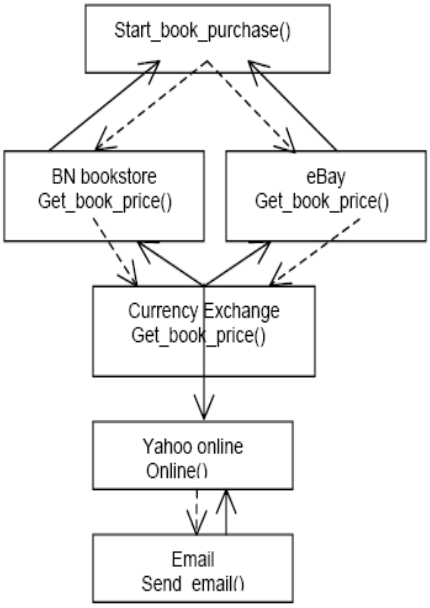

Figure 9. Online Book purchase workflow

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 15 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.

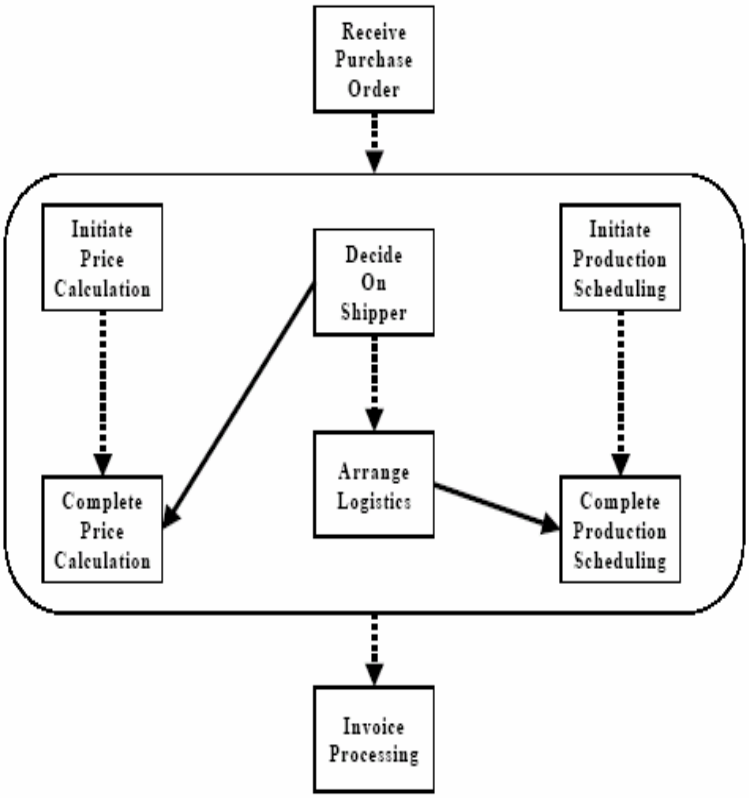

Figure 10. Puchase Order Workflow[32]

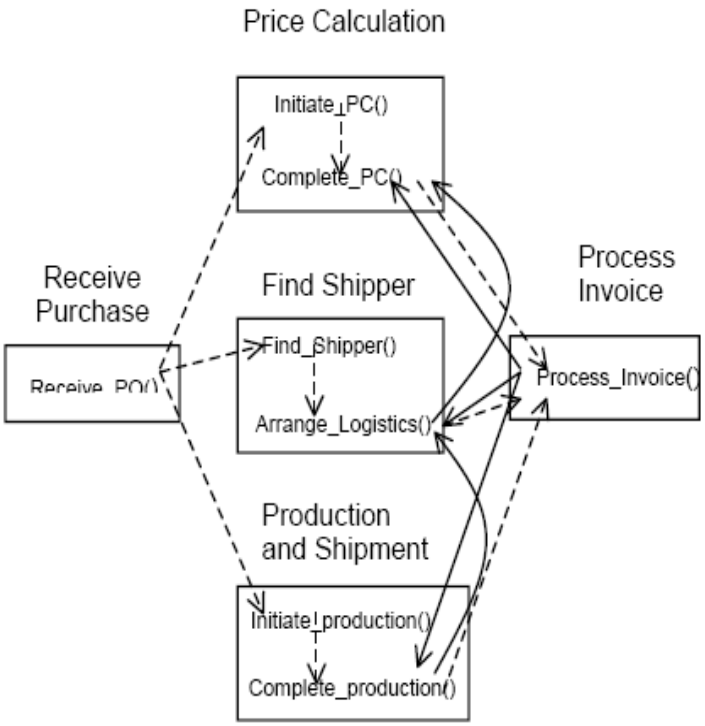

Figure 11. Purchase order workflow modeled using Web Bonds

System performance details: We have deployed and executed these workflows on both wired and wireless infrastructure. Table 1 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 ~5% in wired infrastructure. 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). In case of wireless devices, workflow execution time is relatively high due the time taken by the DOM parser we have used. Total time taken for parsing, accounts for ~35% of the workflow execution time. For this experiment we deployed workflow in a single BondFlow runtime. Performance data for SyD-based distributed workflow will be available for the final submission (currently we are setting up a larger collection of

iPAQs and choosing workflows with better parallelizability. In SyD middleware we have used SyDDoc module to handle XML parsing and time taken is quite small [1]. We can reduce the parsing related time using SyDDoc. In wireless infrastructure, bond related timing accounts for ~30% once we deduct the XML parsing time. Table2 shows the footprints of two workflows. The coordinator objects and corresponding bond repositories, accounts for ~25% and ~75% respectively. Size of the bond repository increases with number of bonds each web service has. Finally, chart 1 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. The following tables give a modular view of the various operations of the system and their performance measures in various configuration environments.

Table 7. Performance Measurements

Head	Distributed Version (SyD Based)	Centralized Version (iPAQ based)	Centralized Version (PC based)
Subscription Bond	583 ms	500 ms	15 ms
Mark And Lock a method	3628 ms	1468 ms	109 ms
Change a method	3773 ms	1476 ms	20 ms
PRE Negotiation Bond (Method not executed)	4863	4529 ms	400 ms
PRE Negotiation Bond (Method already executed)	2299 ms	1175 ms	78 ms

8 CONCLUSION & FUTURE WORK

In the future, we would like to build a complete IDE for BondFlow system. It will have capabilities of web coordination bonds so that developers can enforce all the workflow control flow dependencies and distributed communication patterns. Also, current menu driven system needs to be enhanced towards drag and drop kind of IDE. Further, we need to enhance the runtime engine of our systems so that users can monitor the execution and make changes while workflow is active. Finally, we would like to further investigate the power consumption of workflow execution on small handhelds.

9 BIBLIOGRAPHY

1. Sushil K. Prasad, V. Madisetti, Sham Navathe, et al. System on Mobile Devices (SyD): A Middleware Testbed for Collaborative Applications over Small Heterogeneous Devices and Data Stores, in Proc. ACM/IFIP/USENIX 5th International Middleware Conference, Toronto, Ontario, Canada, October 18th - 22, 2004.

2. 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

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

4. Sushil K. Prasad and Janaka Balasooriya, 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

5. 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.

6. W.M.P. van der Aalst, Workflow patterns, http:/tmitwww.tm.tue.nl/research/patterns, 2003.

7. Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for web service composition. In Proceedings of the Eleventh International World Wide Web Conference, Honolulu, Hawaii, May 2002

8. 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.

9. Alonso, G., Casati, F., Kuno, H., Machiraju, V., "Web Services Concepts, Architectures and Applications Series: Data-Centric Systems and Applications," 2004, Springer, ISBN: 3-540-44008-9.

10. Girish Chafle, Sunil Chandra, Vijay Mann and Mangala G. Nanda. Decentralized Orchestration of Composite Web Services. In Proceedings of the Alternate Track on Web Services at the 13th International World Wide Web Conference (WWW 2004), New York, NY, May 2004.

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

12. 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.

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

14. 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.

15. Portal of NASAs Mars Exploration Rovers Mission,,Elias Sinderson (CSC, NASA ARC, U.S.), Vish Magapu (SAIC, NASA ARC, U.S.), Ronald Mak (RIACS, NASA ARC, U.S.), "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. pp 1-17

16. Anand Ranganathan, Scott McFaddin, Using Workflows to Coordinate Web Services in Pervasive Computing Environments. Proceedings of the IEEE International Conference on Web Services (ICWS'04), June 6-9, 2004, San Diego, California, USA. IEEE Computer Society 2004, 288-295

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

18. Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, 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.

19. Schahram Dustdar, Harald Gall, Roman Schmidt: Web Services for Groupware in Distributed and Mobile Collaboration. PDP 2004

20. 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

21. 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.

22. Hawryszkiewycz, 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.

23. 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.

24. W.M.P. van der Aalst "Don't go with the flow: Web services composition standards exposed. Web Services - Been there done that?, Trends & Controversies", Jan/Feb 2003 issue of IEEE Intelligent Systems

25. Sushil K. Prasad and J. Balasoorya, 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.

26. 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.

27. Janaka Balasooriya, Mohini Padye , Sushil Prasad, and Shamkant B. Navathe "BondFlow: A System for Distributed Coordination of Workflows over Web Services," In 14th HCW in conjunction with IPDPS 2005. Denver, Colorado, USA, April 4.

28. W.M.P. van der Aalst, L. Aldred, M. Dumas, and A.H.M. ter Hofstede. Design and Implementation of the YAWL system. To appear in Proc. of The 16th Intel. Conf. on Advanced Information Systems Engineering (CAiSE 04), Riga, Latvia, June 2004

29. W. van der Aalst and A. Hofstede, Yawl: Yet another work-flow language, 2002

30. Web Service Composition - Current Solutions and Open Problems - Biplav Srivastava and Jana Koehler, IBM Research Lab

31. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, Business Process Execution Language for Web Services version 1.1