Georgia State University
Digital Archive @ GSU

Computer Science Theses | Department of Computer Science

12-20-2004

Coordinating Heterogeneous Web Services through Handhelds using SyD's Wrapper Framework

Mohini Padhye
mohini_padhye@hotmail.com

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

Recommended Citation
Padhye, Mohini, "Coordinating Heterogeneous Web Services through Handhelds using SyD's Wrapper Framework" (2004). *Computer Science Theses.* Paper 1.

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.

Coordinating Heterogeneous Web Services through Handhelds using SyD's Wrapper Framework

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

2004

by

Mohini Padhye

Committee:

Dr. Sushil K. Prasad, Chair

Dr. Anu G. Bourgeois, Member

Dr. Alex Zelikovsky, Member

December 2, 2004
Date

Dr. Martin Fraser

Department Chair

Abstract

Tying web services together to build large, distributed, collaborative applications has gathered noticeable momentum and a lot of research is being put in it. Along with composition of the web services, coordination is one key aspect that has been considered keenly.

Many frameworks, languages and protocols have been proposed for web service composition and coordination. With the advancement in wireless technology and rapid deployment of mobile services, collaborative application development for small devices using such composed web services finds a new research area. Much less work has been done in the area of web service coordination for mobile environment.

In this thesis, we propose a new distributed approach in service composition and coordination and show that our approach works well in an environment containing mobile heterogeneous devices. We discuss a novel approach of SyD (System on Devices) wrapper framework for dynamically creating and executing web bonds [11] among various heterogeneous web services. The wrapper is a lightweight SyD application object that encapsulates composition and coordination logic and provides higher level of coordination among bonded entities. The wrapper framework gives small devices full capability to run distributed collaborative applications that use heterogeneous web services. We have also developed and analyzed experiments to showcase the performance of SyD Wrapper Framework.

Acknowledgements

I would like to thank my advisor, Dr. Prasad, for his ingenious guidance throughout my thesis work. Dr Prasad has always been very kind and supportive in all the work that I carried out. This work would not have been possible without Dr Prasad's vision and insightful suggestions.

I am also very grateful to the member of my thesis committee, Dr. Bourgeois for her help in providing some important reference material.

I would like to thank both the committee members, Dr Bourgeois and Dr. Zelikovsky for spending their valuable time in reviewing this material and offering some useful suggestions.

Dedicated to Aai, Pappa and Rahul

For all the love and support

TABLE OF CONTENTS

Chapter 1 Introduction 1

1.1 Need for Web Service Composition and Coordination 2

1.2 Limitations of Current Technologies 4

1.3 Our Solution 6

1.4 Thesis Organization 7

Chapter 2 Web Services 8

2.1 WSDL 8

2.2 UDDI 9

2.3 SOAP 10

2.4 Issues Involved in Web Service Composition and Coordination 11

Chapter 3 SyD Middleware and Web Coordination Bonds 14

3.1 SyD Architecture 15

3.2 Web Coordination Bonds 18

Chapter 4 SyD Wrapper Framework 19

4.1 System Architecture Diagram 19

4.2 User Interface Module 21

4.3 Web Service Interface Module 23

4.4 SyD Wrapper Generator Module 26

4.5 Implementation Details 29

Chapter 5 Distributed Coordination of SyD Wrapper Objects 31

5.1 System Architecture 31

5.2 Wrapper Registration as a SyD Application Object 32

5.3 Wrapper Invocation through SyD Engine 33

5.4 Comparison between Centralized and Distributed Coordination Approaches . 33

Chapter 6 Experiments and Results 35

6.1 Distributed Travel Application using SyD Wrapper Application Objects 35

6.2 More Example Scenarios for SyD Wrapper Framework 37

6.3 Performance Analysis 38

Chapter 7 Related Work **42**

Chapter 8 Conclusion and future work **45**

8.1 Conclusion 45

8.2 Future Work 46

Bibliography **47**

Appendix A – Wrapper API **52**

Appendix B - Source Code **67**

List of Figures

Figure 1 Simplified WSDL Layout **9**
Figure 2 Simplified Soap Layout **10**
Figure 3 Web Service Technology Stack **11**
Figure 4 Relation between Composition Languages and other WS Standards **13**
Figure 5 SyD Runtime Environment **14**
Figure 6 SyD Architecture **16**
Figure 7 SyD Wrapper Framework Architecture **20**
Figure 8 Web Service Interface **24**
Figure 10 flight Web Service **25**
Figure 11 SyD Wrapper Generator **26**
Figure 12 Bond information in XML Storage **28**
Figure 13 Subscription Bond Scenario **28**
Figure 14 SyD-Wrapper Integration Scenario **31**
Figure 15 Travel application Web Services **36**
Figure 16 Subscription Bond Scenarios **37**

List of Tables

Table 1 Web Service Wrapper Timings **38**
Table 2 Subscription Bond Timings **39**
Table 3 Negotiation Bond Timings **39**
Table 4 Wrapper Registration with SyDDirectory **40**
Table 5 Distributed Workflow Execution Time with SyD **40**

Chapter 1 Introduction

Recently, a lot of attention has been given to the problem of composing autonomous web services, mainly, to achieve interoperability among diverse applications. Service composition has the potential to reduce development time and effort for new applications. The web domain serves as an interesting environment for service composition because several independent services are provided on the web and there is an inherent need for composing complementary services to achieve the end-user's needs [16]. Web services allow an easy integration of heterogeneous systems due to the fact that they are platform and language independent. Some of the important functionalities needed for web service composition and coordination include atomic transaction processing, automatic flow of control and data, constraint satisfaction, dependency enforcement etc. Some of the frameworks emphasize on transaction and workflow processing.

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It uses XML to exchange data (information) with other applications on other computers by using Internet protocols [1]. We will discuss web services in detail in chapter 2. A web service encapsulates the computational complexity and device heterogeneity and the client entities interact through the interface. Even though web services technology is one of the most highly talked technologies, it is still not widely used. This is mainly due to lack of easy to use, reliable, and robust development environment for web service application development.

Our main aim is to effectively compose web services and maintain coordination among them. System on Devices (SyD) middleware offers a chance for our proposed framework

to be used on wireless heterogeneous device network. SyD is envisioned as a middleware that will enable rapid prototyping and implementation of distributed applications that need a collection of heterogeneous, independent databases to collaborate with each other in a mobile environment [5]. Execution of SyD Wrapper in SyD environment and a small experiement for handheld devices using proposed framework is aimed.

We have used *web coordination bonds* analogous to chemical bonds, as a set of core artifacts for effective collaboration among web services. We propose and implement a novel concept of SyD wrappers that hides the composition and coordination logic from users yet providing effective service coordination among heterogeneous web services. The wrapper implements two types of coordination bonds, viz., *subscription bond* that allows information, control and event flows, *negotiation bond* that enforces dependencies and contracts. Several such SyD wrappers can collaborate with each other to build large distributed applications. The wrappers act as SyD application objects and seamlessly fit in the SyD middleware [5] environment. This gives small devices much power to run such collaborative distributed applications with composed web services. Performance evaluation of this approach is carried out in the end.

1.1 Need for Web Service Composition and Coordination

As web services become more prevalent, tools will be needed to help users alter and integrate these services, mainly to achieve interoperability between diverse applications. Reduction in development time and effort is one of the main reasons for service composition. Composing existing services to obtain new functionality will prove to be essential for both business-to-business and business-to-consumer applications.

Web services allow an easy integration of heterogeneous systems due to the fact that they are platform and language independent. Web service system constitutes of three major components to define standards for discovery, description and messaging protocol viz. Universal Description, Discovery, and Integration (UDDI) [4], Web Services Description Language (WSDL) [3] and Simple Object Access Protocol (SOAP) [33]. However, these standards are not capable of composing existing services dynamically.

Web services technologies are emerging as a powerful vehicle for organizations that need to integrate their applications within and across organizational boundaries. In particular, the process-based composition of web services is gaining a considerable momentum as an approach for the effective integration of distributed, heterogeneous, and autonomous applications. Current state of the art in web service composition is to model the composite web service as a separate web process because web services are stateless and not capable of actively participating in such application scenarios.

Coordination is a fundamental requirement of a number of distributed systems, including web services. Currently, there are many languages and standards for Web services composition and coordination [7]. However, the type of coordination protocol that a system uses may vary depending upon the application and underlying system and there is no fundamental, theoretically sound, WS composition/choreography framework [3].

A variety of distributed applications require coordination. Few of the examples include workflow, business-to-business activities, atomic transactions, security etc. All these applications require some level of composition and coordination. Owing to the high

demand, much of work is being done in the field of web service composition, as well as coordination.

Less work has been done in the area of composition and coordination of web services for small handheld devices. [11] talks about web coordination bonds that are analogous to chemical bonds for composing and coordinating web services. They propose a simple yet powerful design for web bonds. Owing to the lightweight code and simple design, the applications built using web bonds can be ported easily to small handheld devices running SyD middleware.

1.2 Limitations of Current Technologies

Complexity is one of the major limitations of many of the proposed composition and coordination protocol. Quite less work has been done in the area of web service composition for small mobile wireless devices. Disconnection and memory constraints are two important issued considered while designing any application targeted for small handheld devices. [6] surveys the issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition. It states that many of the current technologies, still, do not cover all these aspects in their implementation. Some of the proposed approaches which deal with centralized coordination of web services suffer from central point of failure despite of making the design and implementation simple. Achieving coordination in collaborative applications consisting of composed web services for mobile environment is still an evolving area and much work needs to be done.

1.2.1 Service Compositions Standards

BPEL4WS is highly talked standard and is said to combine the best standards for Web services composition, such as IBM's WSFL and Microsoft's XLANG. XLANG is a block-structured language with basic control flow structures for service composition. [24]. In contrast to XLANG, WSFL (Web Service Flow Language) is not limited to block structures and allows for directed graphs [14] for composing the services. Though WSFL offers more functionality, it adds considerable complexity from the developer's point of view. BPEL4WS allows for a mixture of block and graph structured process models, thus making the language expressive at the price of being complex [26]. 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. An abundant number of languages/standards still failed to give a framework which was fundamentally sound and yet powerful in operation. To overcome this problem, initially, a critical evaluation of these standards is required.

[26] argues that current composition languages are not mature enough to be applied. For example, BPEL4WS mainly focuses on static composition on the service where process flow and the bindings between services are known in advance. It lacks the dynamic binding. Chapter 7 discusses current technologies and their limitation is detail.

1.2.2 Execution of composed services in mobile environment

Some of current web service composition and coordination architectures inherently assume that services are resident on the wired infrastructure. Few of the examples are eFlow[28], CMI[30], Ninja service Composition architecture [29]. They assume the

services to be connected to each other through stable wired network and reside on nodes or devices that are connected to each other over high bandwidth communication channels [6]. Mobile environment poses some inherent limitations such as disconnection, node failure etc, for running collaborative applications. [6] describes the issues related to service composition in mobile environments and evaluate criteria for judging protocols that enable such composition.

1.3 Our Solution

We have proposed and implemented an efficient distributed framework, SyD wrapper, for web service composition which provides high level of coordination using web coordination bonds [11]. This is a unique of its kind framework, which considers all the important constraints that the mobile environment presents. Wrapper is a java class instance that contains the original web service calls along with web service coordination intelligence. The wrapper methods possess exact signature as the web service methods and coordination logic is transparent to the user. For a user, calling the wrapper method is same as calling the original web service method. Wrapper creation is a very simple process.

Web coordination bonds provide a set of core artifacts for Web Service coordination/choreography. They allow rapid modeling and deployment of collaborative applications of all kinds and complexities. The novel approach of SyD wrapper that implements SyD bonds acts as SyD application object and works with SyD middleware [5] for developing and executing collaborative distributed web applications based on composed services in mobile heterogeneous environment. The wrappers are generated

corresponding to web services of interest, with added functionality for service coordination. The wrapper execution is distributed and dynamic.

Much less or no research has been done in the area of web service composition and the collaborative application creation for small handheld devices using composed web services. The current technology for deployment of collaborative application over heterogeneous set of wired or wireless devices and networks has several limitations. The SyD's wrapper framework helps overcome this problem by great means. Chapter 5 discusses the coordination of SyD wrapper objects in detail. We have carried out an experiment to showcase the working of SyD Wrapper Framework. The experiment is discusses and results are analyzed in subsequent chapters.

1.4 Thesis Organization

This document is organized in 7 chapters. Chapter 2 introduces Web Services and discusses issues involved in Web Service Composition and Coordination. Chapter 3 gives an overview of SyD Architecture and Web Coordination Bonds. Chapter 4 presents the SyD Wrapper Framework and gives the implementation details. The process of achieving coordination among heterogeneous web services in a mobile environment is presented in chapter 5. Chapter 6 discusses the experiments and analyzes the results. Chapter 7 talks about the related work in the area of Web Service Composition and Coordination for wired as well as mobile environment. Conclusion and future work is discusses in chapter 8. Bibliographical references follow chapter 8. APIs used and Source code forms a part of appendices.

Chapter 2 Web Services

A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web service interfaces are defined using WSDL and other systems interact with the web service using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards. [1].

Web Services combine the best aspects of component-based development and the World Wide Web. They are considered as software applications that use XML to exchange data/information with other applications on other computers by using Internet protocols. Web services operate over any network (the Internet or a private intranet) to achieve specific tasks, called methods or functions, which other applications can invoke and use [2]. Requests can be sent and responses received between two differing applications on two separate computers belonging to separate business enterprises or small businesses. A large range of application domains starting from small businesses to global enterprises benefit enormously from web Services.

The next subsections discuss three main components of a web service system, viz., WSDL, UDDI and SOAP.

2.1 WSDL

Web Services Description Language (WSDL) provides XML format for describing web services. It is an XML-based language that is used to describe the services the web service offers and to provide a way for identifying and accessing those services

electronically. WSDL enables one to separate the description of the abstract functionality offered by a service from concrete details of a service description [3].

```
<definitions>
  <types/> ...            describes the datatypes in XML
  <message/> ...          describes messages (name and parameter)
  <portType> ...          describes the callable messages for each protocol. The messages
    <operation>           are grouped together to an operation.
      <input/>            Thus, this elements describe these operations which can be
      <output/>           reached via this "port".
    <binding> ...         binds a protocol to a port and defines the used encoding
      <operation/>        for each operation (e.g. SOAP, rpc/encoded).
    <service>       ...   describes the identification of a service: name, used bindings
      <port/>             and the URI under which the service can be reached
</definitions>
```

Figure 1 Simplified WSDL Layout

2.2 UDDI

The Universal Description, Discovery and Integration (UDDI) acts as a registry service for web services. A UDDI registry service is, basically, a Web service that manages information about service providers, service implementations and service metadata. The web service providers can use UDDI to advertise the services they offer. Service consumers can use UDDI to discover services that suit their requirements and to obtain the service metadata needed to consume those services. [4]

2.3 SOAP

Simple object access protocol (SOAP) is a communication protocol for applications running on different operating systems, with different technologies and programming languages.

[1] defines SOAP as an XML based lightweight protocol for exchange of information in a decentralized, distributed environment. SOAP consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined data types, and a convention for representing remote procedure calls and responses [1].

The following figure shows a simplified layout of SOAP. It shows two important parts of a SOAP document, viz., header and body. SOAP envelope is the element that encapsulates the header and body.

```
<SOAP:Envelope
  xmlns:SOAP='http://schemas.xmlsoap.org/soap/envelope/'
  SOAP:encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'
  Xmlns:v='http://www.topxml.com/soapworkshop/'>
  <SOAP:Header>
    <v:From SOAP:mustUnderstand='1'>
      cdix@soapworkshop.com                SOAP Header
    </v:From>
  <SOAP:Header>                                          SOAP Envelope
  <SOAP:Body>
    <v:DOCreditCheck>
      <ssn>123-456-7890</ssn>              SOAP Body
    </v:DOCreditCheck>
  </SOAP:Body>
</SOAP:Envelope>
```

Figure 2 Simplified Soap Layout

The popularity of SOAP lies in its platform independence. It provides a means for accessing disparate services, distributed objects and servers in a totally platform-independent manner [32].

The Web services platform comprises different kinds of technologies and standards that are organized into the five layers of network, transport, packaging, description and discovery, as illustrated in Figure 3.

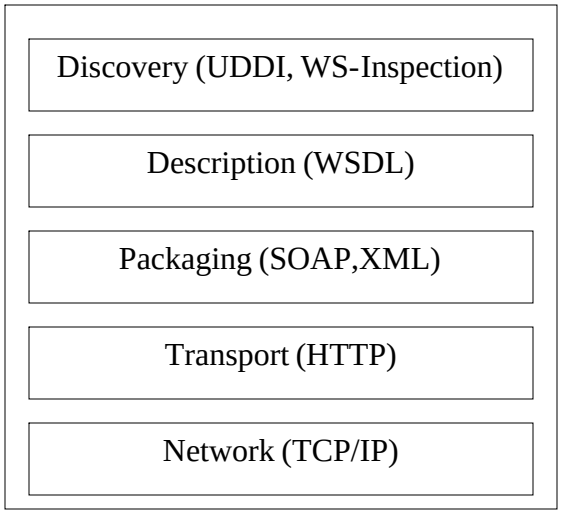

Figure 3 Web Service Technology Stack [10]

2.4 Issues Involved in Web Service Composition and Coordination

Some of the issues involved in web service composition and coordination are discussed in this section. Various standards have been proposed for service composition. Some of the important issues that are considered while designing composition standard, language or protocol are as follows,

- **Type of composition**

  There are two types of process composition: static and dynamic. In a static composition, the services to be composed are chosen at design time, while in a dynamic composition, they are chosen at run-time.

- **Nature of composition**

  Compositions can be of two types in nature, viz., centralized and distributed. The designer has to consider the nature of composition prior to actual design.

- **Specification of services**

  Service specification should be studied in detail before going ahead with the composition of services.

- **Interoperability issues**

  Achieving interoperability among various components and resolving interoperability issues should be given prime importance.

- **Type of coordination (process execution)**

  The way coordination is achieved among various components of the composed web services is mainly based on the way the web services are composed. So composition and coordination go hand in hand.

- **Application domain**

  Application domain is always studied keenly before going ahead with composition and coordination of various components in the system.

Figure 4 shows relationship between web service composition languages and other standards such as SOAP, WSDL and UDDI.

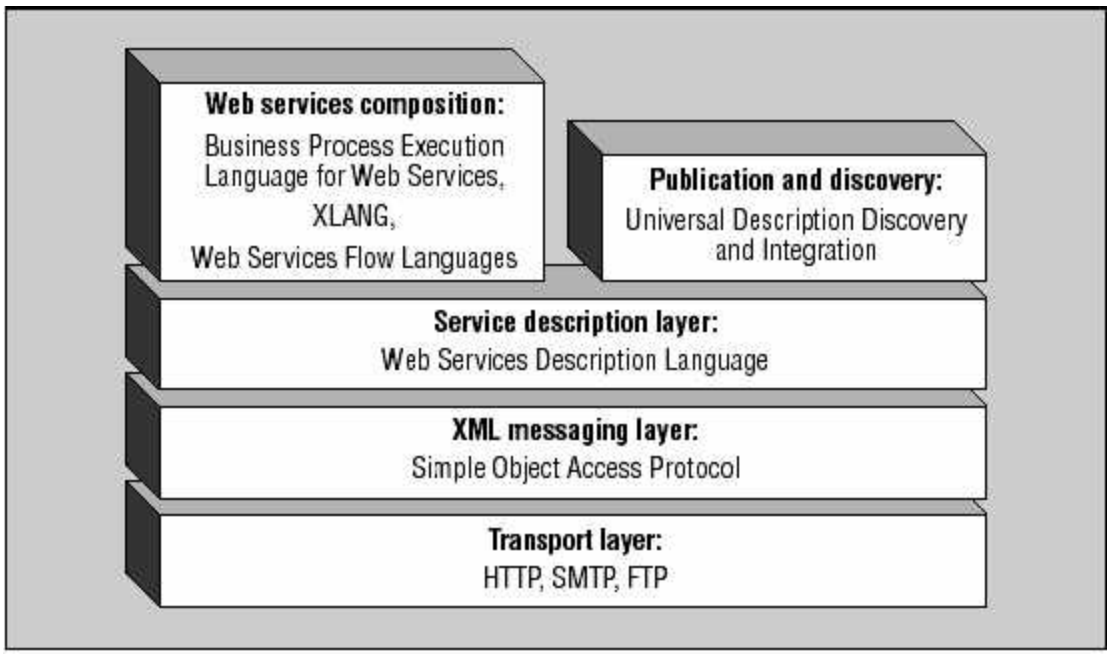

Figure 4 Relation between composition languages and other WS standards [26]

Chapter 3 SyD Middleware and Web Coordination Bonds

System on Devices (SyD) is envisioned as a middleware that will enable rapid prototyping and implementation of distributed applications that need a collection of heterogeneous, independent databases to collaborate with each other in a mobile environment. This middleware can be installed on traditional computers, laptops, PDAs or any other heterogeneous computing device. SyD acts as a rapid application development platform and SyD-based application development greatly reduces the development, implementation, deployment, and maintenance time for designers and programmers of distributed applications on heterogonous mobile devices and environments [5].

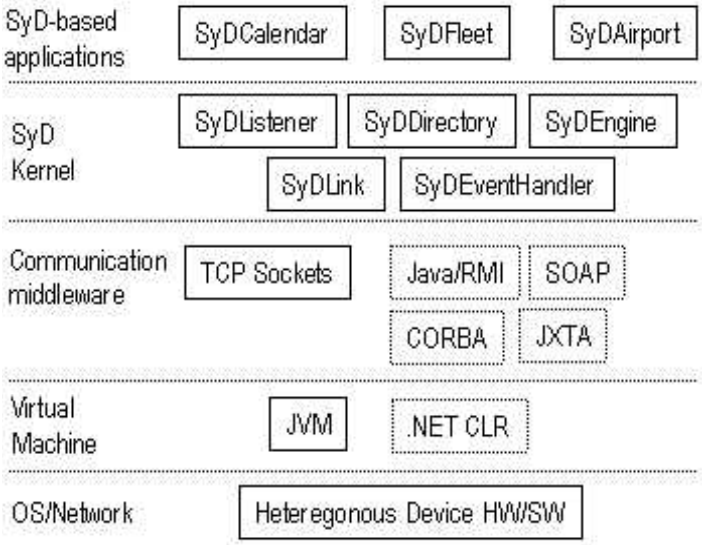

Figure 5 SyD Runtime Environment [18]

Figure 5 shows SyD runtime environment where SyD middleware is located between applications and the communication services provided by primitive distribution middleware (Sockets, RMI, JXTA, CORBA, etc). The figure also shows applications

such as SyDFleet, SyDCalendar that have been developed over SyD. SyD allows rapid development of a range of portable and reliable applications.

SyD addresses the key problems of heterogeneity of device, data format and network, and that 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.

3.1 SyD Architecture

The SyD Kernel includes following 5 modules, as shown in Figure 6 [5]

❖ **SyDDirectory**: Provides user/group/service publishing, management, and lookup services to SyD users and device objects. Also supports intelligent proxy maintenance for users/devices. SyDDirectory Service

(i) Is based on web service framework (UDDI, WSDL)

(ii) Keeps track of application objects and their associated devices

(iii) Maintains directories of peers and services

(iv) Delivers location information of peers on the fly

(v) Provides distribution transparency

(vi) Communicates through XML-based information

❖ **SyDListener**: The SyDListener has a distributed architecture, with a registration component and a listening part located at a device that can provide services in the SyD network, and a delegate component at a device that will request services from the server device. The delegate acts as a local proxy of the listener object on the server device. This

architecture has the advantage of hiding invocation details of the service from the service requester. This module consists of

1. SyDRegistrar: SyDListener performs registration for service provider through SyD Registrar. It registers objects of SyD applications as remote services.

2. SyDListener: Constantly listens for a service request message from clients of services, parses the message, performs actual method invocation, and returns invocation result as response message Local Event Handler.

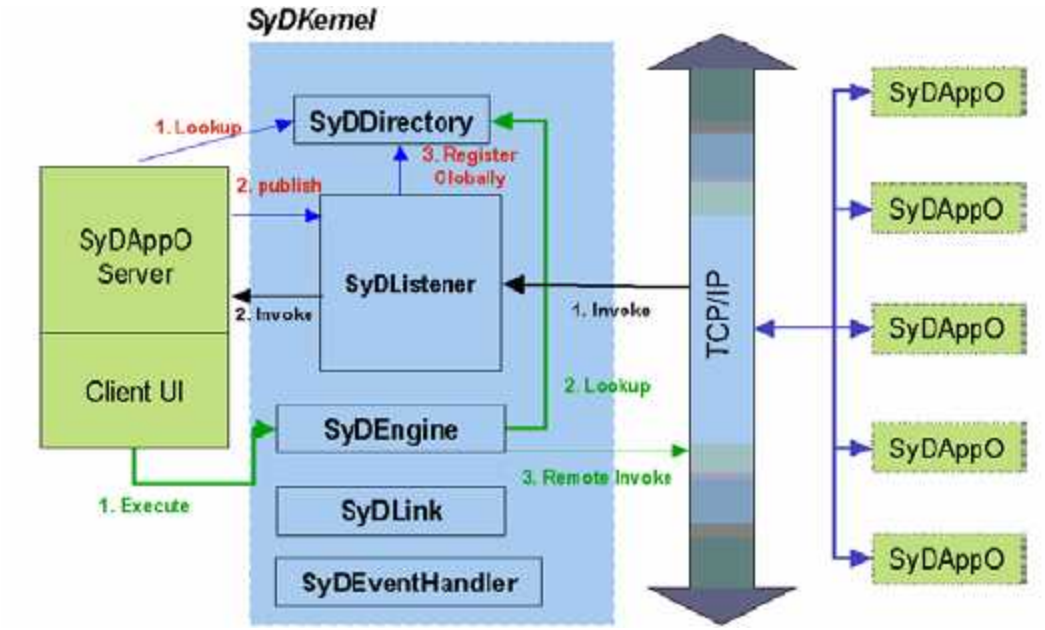

Figure 6 SyD Architecture [18]

3. SyDListenerDelegate: The proxy of SyDListener at client side and performs communication with SyDListener.

❖ **SyDEngine**: Allows users to execute single or group services remotely via SyDListener and aggregates results. This yields a basic composer of mobile web services. It thus,

(i) Provides the mechanism for applications to access the data and other resources at remote devices in a transparent manner

(ii) Is responsible for aggregating results obtained from method calls and presenting the summarized information back to the application

(iii) Manages the proxy when the device is off line

❖ **SyDLink**: Enables an application to create and enforce interdependencies, constraints and automatic updates among groups of SyD entities and Web Services.

❖ **SyDEventHandler**: This module handles local and global event registration, monitoring, and triggering.

(i) Allows Objects to identify state changes that could be of interest to other objects anywhere on the network.

(ii) Allows registration of interest in those state changes.

(iii) Sends notifications when a state change occurs to all who have registered interest.

(iv) Provides fault-tolerance using time-out and proxy

❖ **SyDDoc**: The SyDDoc utility API provides a uniform data exchange capability for SyD middleware and SyD-enabled modules. It is based on XML and is lightweight.

(i) SyDDoc provides support for various data types.

(ii) PublisherDoc is similar to WSDL for publishing services to the Directory Service.

(iii) Invocation framework is similar to Request-Response method used in SOAP over HTTP.

3.2 Web Coordination Bonds

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 [11].

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. 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), status (confirmed/tentative), references to one or more Web entities, triggers associated with each reference , a priority, a constraint (and, or, xor), a bond creation time and a bond expiry time, and a waiting list of tentative bonds (a priority queue). A tentative bond may become confirmed if the awaited confirmed bond is destroyed.

Web bonds are simple yet powerful, and [11] demonstrates how they can be employed to create (model) and enforce (deploy and execute) producer-consumer and shared-resource relationships, workflow scenarios, and atomic transactions.

Chapter 4 SyD Wrapper Framework

Less work has been done in the area of web service composition for small mobile wireless devices. Achieving coordination in collaborative applications consisting of composed web services for mobile environment is still an evolving area. Some of current web service composition and coordination architectures inherently assume that services are resident on the wired infrastructure. Many of proposed architectures are centralized and consist of preconfigured settings for composing as well as coordinating the web services. A distributed framework for service coordination in mobile environments is needed that takes into consideration the constraints associated with mobile environment. We have proposed a SyD Wrapper Framwork that helps achieve this. This section discusses generation of such wrappers which work in the mobile environment. These wrappers can be configured in a distributed manner to run in SyD environment to achieve coordination among heterogeneous web service.

This section discusses the SyD Wrapper Framework to the finest detail. The following subsections give an idea about the Framework on the component basis.

4.1 System Architecture Diagram

The system is divided in three important components viz, UI Module, WS Interface Module and Wrapper Generator module as shown in figure 7. The WS Interface module uses SOAP and UDDI to locate the web service as shown. Web Bond Manager that deals with bond creation and execution works is a part of the wrapper generator module. The figure also shows the SOAP call to original web services.

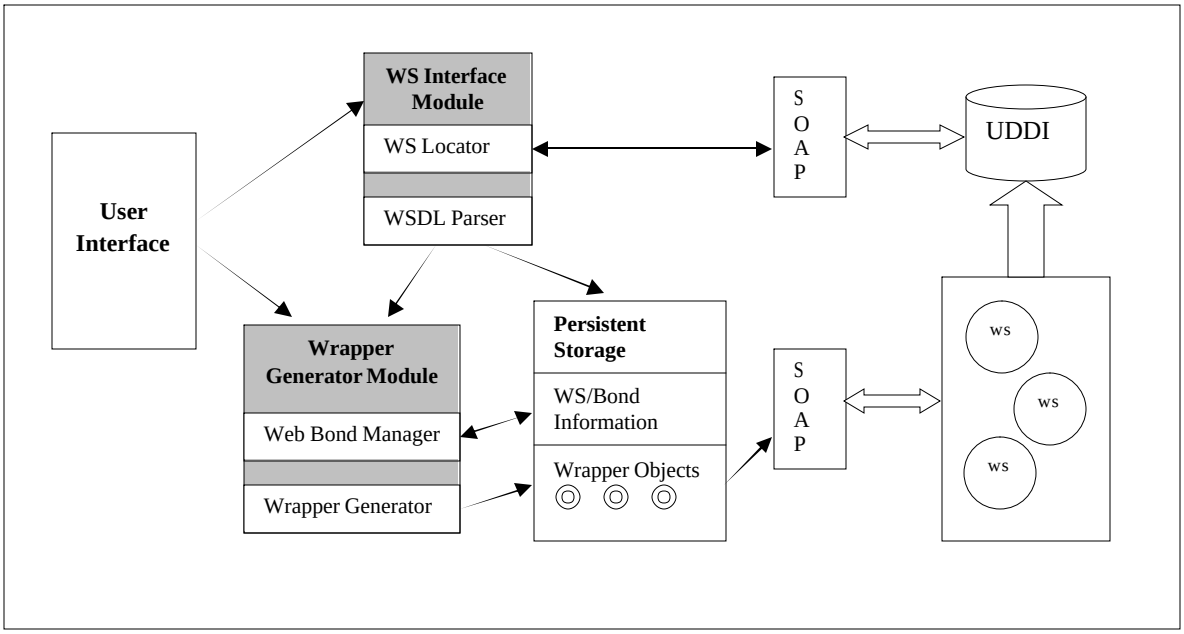

Figure 7 SyD Wrapper Framework Architecture

The SyD Wrapper Framework initiates its operation by web service lookup and discovery. Web service (WS) interface module that contains WS locator helps discovering the service of interest. WSDL Parser parses the WSDL 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. Unlimited number of services can be browsed and saved by the user. Instance of web service wrapper is created when user wishes to save the web service. XML files serve the purpose of permanent storage for the services.

Web coordination bonds can be created among the saved services at any point of time. 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 the XML storage file.

The wrapper encompasses all the coordination capabilities of the web bond artifacts. It hides the heterogeneity of various objects including legacy web services distributed among the network by enabling them to coordinate using SyD Wrapper Framework. The bond coordination logic that the wrapper contains is transparent to the user at all the times. Once the wrapper is created and bonded, the basic skeleton of web service composition for SyD Wrapper Framework is ready.

The wrapper generation process can be centralized or distributed. Bond execution process, though, is distributed. Footprint of the wrapper is small and can reside on a mobile device easily. The wrappers can communicate with each other using SyDListener [5] component of SyD middleware. The SyD Wrapper Framework makes the collaborative application development very easy for small devices running SyD and we will talk about it in detail in the next section.

Once any of the wrappers is invoked, the presence of web bond is initially checked and depending upon the presence and type of bond, coordination among components is carried out by enforcing the specified constraints and dependencies.

The following subsections discuss each component of our system in detail.

4.2 User Interface Module

User Interface Module facilitates the user to locate and discover web services and save the desired services for future reference. The interface allows users to create, delete, update and view the web coordination bond.

The tasks are briefly discussed bellow,

(i) **Search Web Service**: The user inputs the WSDL url for the web service on the user interface. The url that the user inputs is sent to the WSInterface Module which deals with contacting UDDI and fetching the WSDL.

(ii) **View WSDL Details**: This interface enables the user to view a user friendly version of the parsed WSDL which shows list of methods the service contains. It also shows input and return parameter for each method.

(iii) **Save Web Service**: If the user wishes to store the web service to permanent storage for future references, he can do so through this option.

(iv) **View Web Services**: The user gets the opportunity to view the saved web services by selecting this option. It lets the user view the list of all the web services, along with methods and parameters.

(v)**Create Bond**: The user can create web coordination bond through 'Create Bond' option. The parameters he provides for create bond process are the source web service, source method, destination web service, destination method, type of bond (which can be either subscription or negotiation) and presence of trigger (if there is a trigger associated with the bond). The backend function

(vi) **Delete Bond**: This option enables the user to delete the web bond

(vii)**Update Bond**: Any of the bond parameters can be updated using this option.

(viii) **View Bond**: Details of a particular bond can be viewed using View Bond option.

4.3 Web Service Interface Module

This module contains two components, Web Service Locator and WSDL Parser, as shown in Fig 5. 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 also interacts with the XML Storage and the Wrapper Generator module. Following subsections discuss the component of this module in detail

4.3.1 Web Service Locator

When a user wishes to browse through the services provided by a provider, he supplies the WSDL url of that service. The Web Service Locator module locates the service by contacting UDDI, gets the WSDL and passes it to the WSDL Parser module. We have used Apache- Axis implementation of the web services. Figure 8 shows the component level view of the Web Service Locator.

4.3.2 WSDL Parser

WSDL parser uses WSDL4J API for WSDL parsing. It parses the WSDL file for required components. It stores the result in the XML Storage for persistence, if the user opts to save the web service. Some of the entities saved to the storage are the name of the service, locationURI, namespaceURI, methods and parameter list. NamespaceURI and locationURI are used for making a call to the actual web service from the wrapper body.

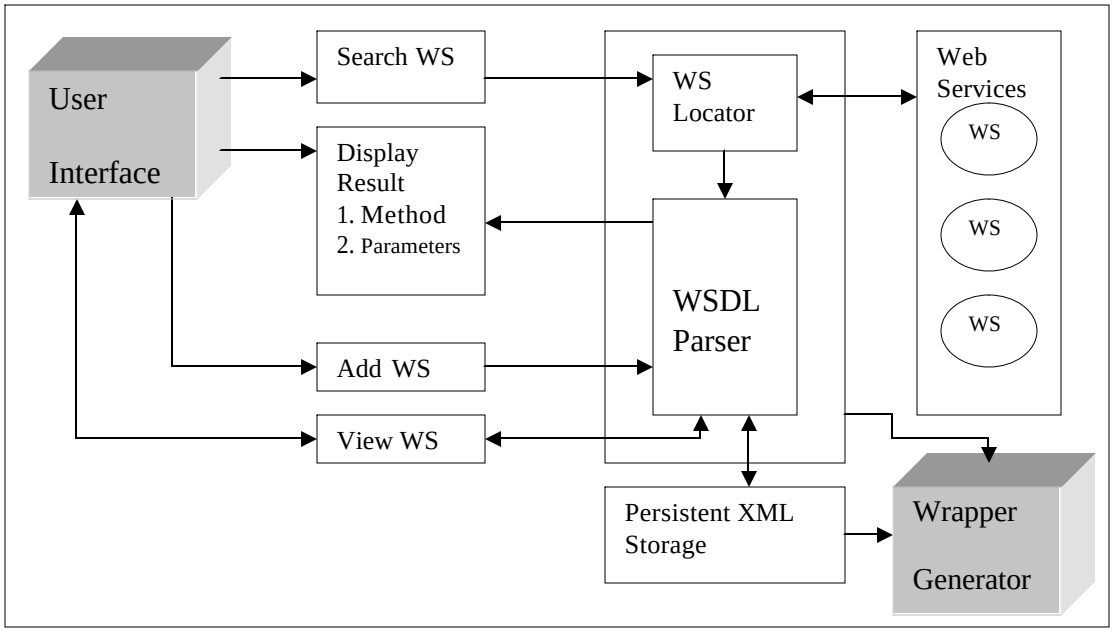

Figure 8 Web Service Interface

XML data is stored in the persistent storage in a predefined format. The important parameters that are stored in the XML file name of the web service, method names along with the list of input and output parameters, location and namespace uri. NamespaceURI and locationURI are used while making the actual call to the web service. A code snippet showing flight web service data stored in XML storage shown in figure 9, (e.g. flight.xml file) and corresponding web service description is shown in figure 10.

Methods and parameter list is shown to the user for his reference and is saved in the XML file for future calls to the web service through the wrapper. Web Service Interface module coordinates between User Interface and Wrapper Generator module.

```
<WebService>
   <WsdlUrl>flight.wsdl</WsdlUrl>
<!-- Name of the web service -->
   <WSName>flight</WSName>
<!-- LocationURI and NamespaceURI are used for SOAP call to the original web service -->
      <LocationURI>http://localhost:8080/soap/servlet/rpcrouter</LocationURI>
   <NamespaceURI>urn:flight</NamespaceURI>
<!-- Method details for each method in the web service. This contains method name, parameter
names along with their data type and return type -->
   <Method mid="1">
        <MethodName>addFlight</MethodName>
       <ParamType>string fno</ParamType>
       <ParamType>string city_from</ParamType>
        <ParamType>string city_to</ParamType>
        <ParamType>integer seat_max</ParamType>
        <ReturnType>string return</ReturnType>
   </Method>
   <Method mid="2">
        <MethodName>cancelFlight</MethodName>
       <ParamType>string fno</ParamType>
        <ParamType>integer seats_num</ParamType>
        <ReturnType>string return</ReturnType>
    </Method>
</WebService>
```

Figure 9 Web Service date in XML storage

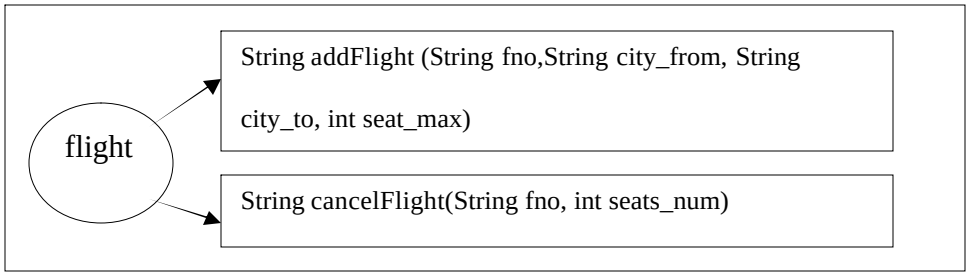

Figure 10 flight Web Service

4.4 SyD Wrapper Generator Module

Two main components of this module are Wrapper Generator and Web Bond Manager. This module forms the heart of the system. It deals with creation of the wrapper object corresponding to the web service and management and execution of web bonds. Figure 11 gives the component view of SyD Wrapper Generator module and next two subsections give a detailed overview of the components of this module.

4.4.1 Wrapper Generator

Wrapper generation per service is carried out by this module. Wrappers encapsulate original web service methods along with bond management functionality. The entire operation of bond management is transparent to the user.

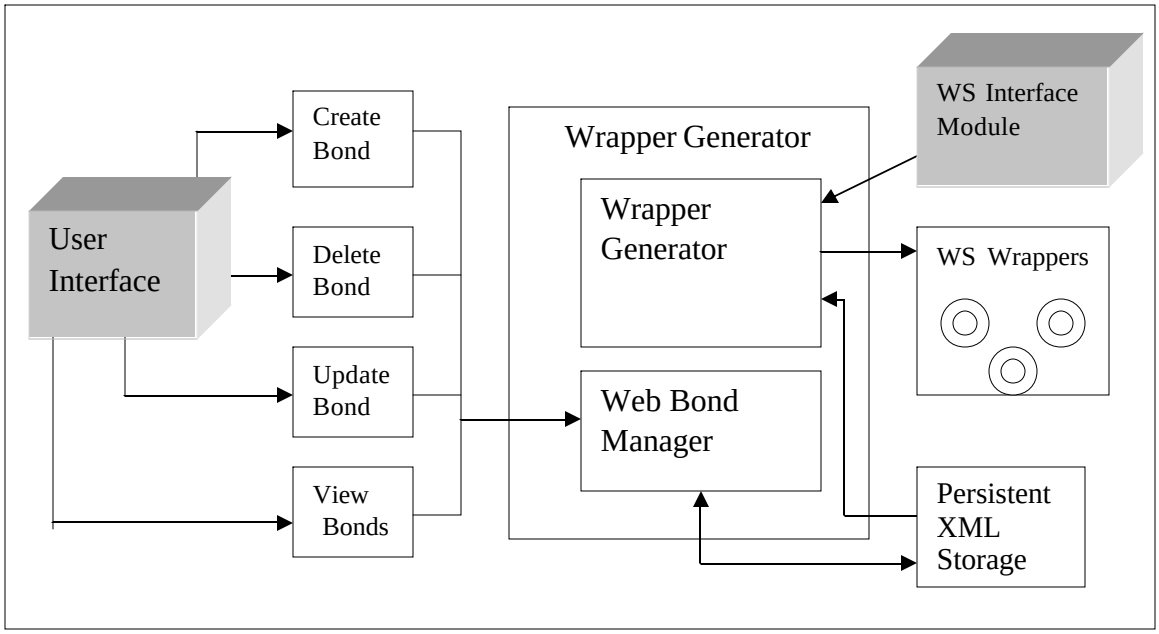

Figure 11 SyD Wrapper Generator

The wrapper is a java class and wrapper methods posses exactly the same signature as that of the original web service methods. For the user, there is no difference in calling the wrapper than the original web service.

The size of wrapper is very small (approximately 11KB for a web service with 6 methods) and it can be easily stored on a handheld device.

The wrapper stores the logic for bond management and has the intelligence of managing the bonds dynamically and enforcing dependencies to cater to the need of a particular coordination scenario.

4.4.2 Web Bond Manager

Web bond manager deals with all the bond related operations, such as creation, deletion and updating of the web bonds.

When the bond is created, bond related information is stored in an XML Storage for future references. Bond parameters are specified while creating the bonds and necessary information is stored in same XML storage. Type of bond created acts as one of the important parameters along with the others. The runtime execution of the bond depends upon the type specified. The XML file is referred upon bond invocation and subsequent operations are carried out according to the bond data.

Upon wrapper invocation, the wrapper consults the Bond Manager and carries out series of operations depending upon the bond parameters specified at the bond creation time. Checking of type of bonds, getting bond parameters and executing the actual bond are some of the major operations by the Bonds Management System. The final call to the original web service is made by the wrapper using SOAP.

Figure 12 shows a code snippet (which is part of flight.xml) that gives an overview of data that is stored at bond creation and figure 13 gives pictorial representation of the same. It shows the bond scenario between addFlight method of flight web service and addHotel method of hotel web service.

```
<Wrapper>
   <WSName>flight</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="1">
      <SrcMethod>addFlight</SrcMethod>
      <DestWS>hotel</DestWS>
      <DestMethod>addHotel</DestMethod>
      <Type>S</Type>
      <Trigger>Y</Trigger>
   </Bond>
</Wrapper>
```

Figure 12 Bond information in XML Storage

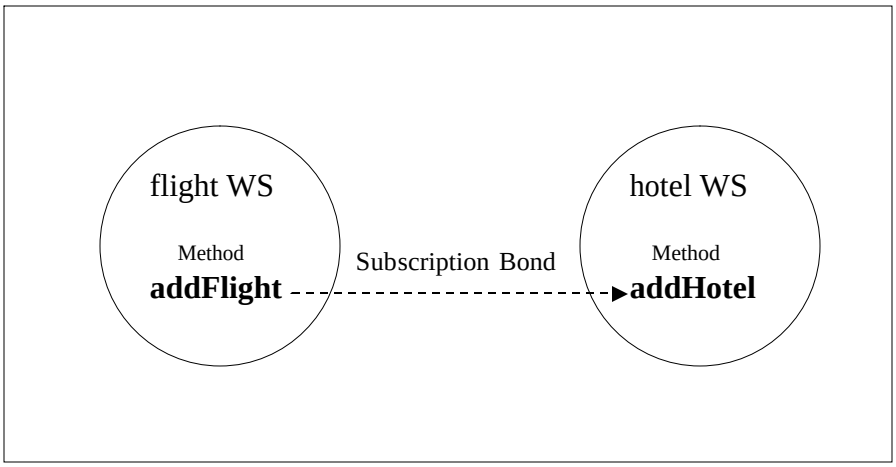

Figure 13 Subscription Bond Scenario

As shown above, the important parameters stored in the persistent storage are source and destination web service name, method name, type of bond (S- Subscription, N-Negotiation) and presence of trigger in this bond.

4.5 Implementation Details

- **Languages, Software and APIs Used**

JDK1.4.1- The Wrapper Generator system is implemented using Java. This makes the system platform independent and provides all the advantages of a typical java-built system.

WSDL4J API [32] - The WSDL parser has been built using WSDL4J API. WSLD4J API is an IBM reference implementation of the JSR-110 specification (Java API's for WSDL). It facilitates the efficient creation, representation and manipulation of WSDL documents which describe the services.

The Web Services Description Language (WSDL) which is an XML-based language for describing web services allows developers to describe the inputs and outputs to an operation, the set of operations that make up a service, the transport and protocol information needed to access the service and the endpoints via which the service is accessible. WSDL4J is a very efficient API for parsing and representing a WSDL document in JAVA.

NanoXML 2.2.1 [31] – Data persistency is achieved in our system using XML storage mechanism and a lot of data manipulation is required during bond creation and execution. A lightweight parser for XML was needed which is exactly what NanoXML provides. A

tree based parser is not efficient to use with small devices due to memory constraints. Event driven parsers like NanoXML are best to use. NanoXML is a small (about 6K), reasonably fast, non-validating XML parser for JAVA that provides a set of APIs.

Axis 1.1 [34] – Locating the web service and subsequent communication with the web services is implemented using Apache Axis which is essentially a soap engine. It provides many other important features along with extensive support for WSDL and compatibility with tomcat. Axis offers improved speed, flexibility and stability. Axis, along with apache web server, provides an efficient web service interface for various applications.

Other softwares that are used are Oracle 8i, Apache 2.0.52, Jakarta Tomcat 5.0.

Chapter 5 Distributed Coordination of SyD Wrapper Objects

The main goal of the SyD Wrapper Framework discussed above is to be able to work with small devices. With the advancement in wireless technology and greatly increased usage of small wireless devices, it has become important that the user of such devices should also be able to avail all important facilities that are provided to the wired user. SyD middleware which is especially built for rapid development of collaborative applications over small devices can help attain this.

5.1 System Architecture

SyD wrappers can be converted into SyD application objects sitting on top of the middleware. Figure 14 shows the view of distributed coordination among various SyD wrapper objects.

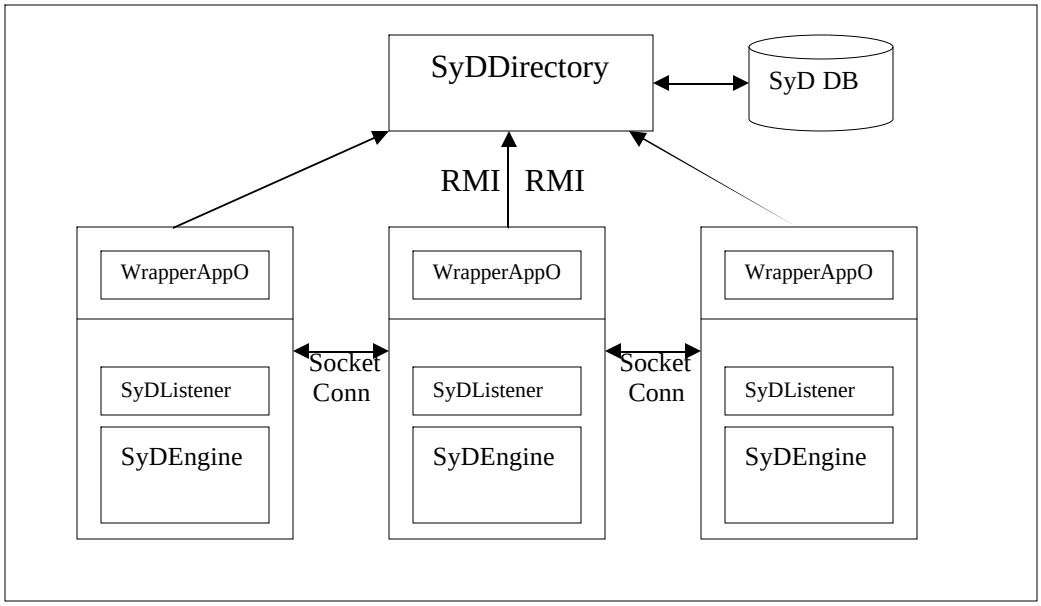

Figure 14 SyD-Wrapper integration scenario

The SyD wrapper objects now acting as SyD application objects take advantage of all the functionalities that the middleware provides. Different SyD application objects can communicate with each other through SyDListeners.

SyDDirectory lets all the wrappers register themselves with the directory through SyDRegistrar so that other wrappers can lookup for them. 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 wrappers can lookup for remote wrappers through SyDDirectory.

SyDEngine facilitates the wrappers to actually invoke a remote wrapper acting as a SyD application object and communicate with it. SyDListener keeps listening for any connection requests and delegates the control to SyDEngine module.

5.2 Wrapper Registration as a SyD Application Object

The flexibility of SyD and scalability of SyD Wrapper framework easily allow both the entities to be integrated. Wrappers can act as SyD application objects running at the application layer in the network hierarchy of SyD middleware.

For the wrappers to be working as SyD objects and communicating with each other, registration with the SyDDirectory is an important operation. Once registered with the SyDDirectory, other wrappers can easily lookup for them and communicate once found.

The wrapper registers 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 registration process with the SyDDirectory begins. Once bound

in the registry, these wrappers wait for invocation from other wrappers. In this scenario, the registered wrappers act as servers waiting for invocation from clients.

5.3 Wrapper Invocation through SyD Engine

Wrapper objects can invoke other wrappers with who they share SyD bond. We have created a subscription bond scenario and will discuss that in detail in the next section. When a collaborative application containing SyD wrapper application objects encounters presence of web bond with other application, then, it looks up for the desired web service in the SyDDirectory. 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 which in turn calls the server's SyDListener by opening a socket connection. The result is returned to the client as an XML document and the communication continues.

5.4 Comparison between the Centralized and Distributed Service Coordination Approaches

The coordinating entities can reside at the centralized as well as distributed locations. A similar approach of web service coordination for handhelds is discussed in [35]. It presents a way of coordination where the coordinating entities reside on a single centralized server. The coordination bonds are created among the interested entities using SyDLink component of the SyD Middleware. After the bonds are created accoding to the business logic, the coordination framework is ready. The coordinating entities are

assumed to be residing on the central location and the communication takes place accordingly. [35] presents a similar experiment of travel application in a centralized execution scenario. The biggest disadvantage of this approach is central point of failure. If the server hosting the coordinating entities fails due to some means, then the entire operation of coordination fails. Also, in real world scenarios, this approach finds a very little scope.

On the other hand, distributed coordination finds many real time applications. Because, the coordinating entities are distributed all over the network, even if one of the device fails, only entities dependent on it will be affected and other operations can work seamlessly. [6] compares in detail the centralized and distributed approaches in service coordination

Chapter 6 Experiments and Results

To showcase the strength of SyD Wrapper objects, we created a dummy travel application scenario and conducted some performance tests on it.

Initially, we, developed and deployed three web services, viz., flight, hotel and car on the test machine We created three wrappers corresponding to these web services and created some dummy bonds among the methods of those wrappers.

These wrappers were, then, registered as SyD application objects and working of the wrappers in the SyD environment is analyzed.

6.1 Distributed Travel Application using SyD Wrapper Application Objects

Initially, we setup a web service scenario containing three web services corresponding to flight, hotel and car services. We created subscription bond between cancelFlight and cancelHotel services of flight and hotel web services respectively and cancelFlight and cancelCar services of hotel and car web services respectively. This means, whenever cancelFlight method on the flight web service is executed, cancelHotel method of hotel web service should automatically be invoked. Similarly, it adds dependencies between flight and car web services bonding cancelFlight and cancelCar method, by invoking cancelCar method if cancelFlight method is invoked. Other methods, such as reserveFlight, reserveHotel and reserveCar of flight, hotel and car services are not bonded and can be invoked without any dependencies. Figure 15 shows this scenario in detail.

Few more subscription bonds are created for more analysis. Figure 16 shows the entire scenario for all the bonds.

The wrappers, thus created per service, bear necessary coordination information. The wrappers are self-sufficient and contain all the bond related data that is used while executing the bonds. Once the scenario is built and wrappers are created, they are ready to be integrated with SyD. A travel application scenario created can be ported to small devices running SyD middleware.

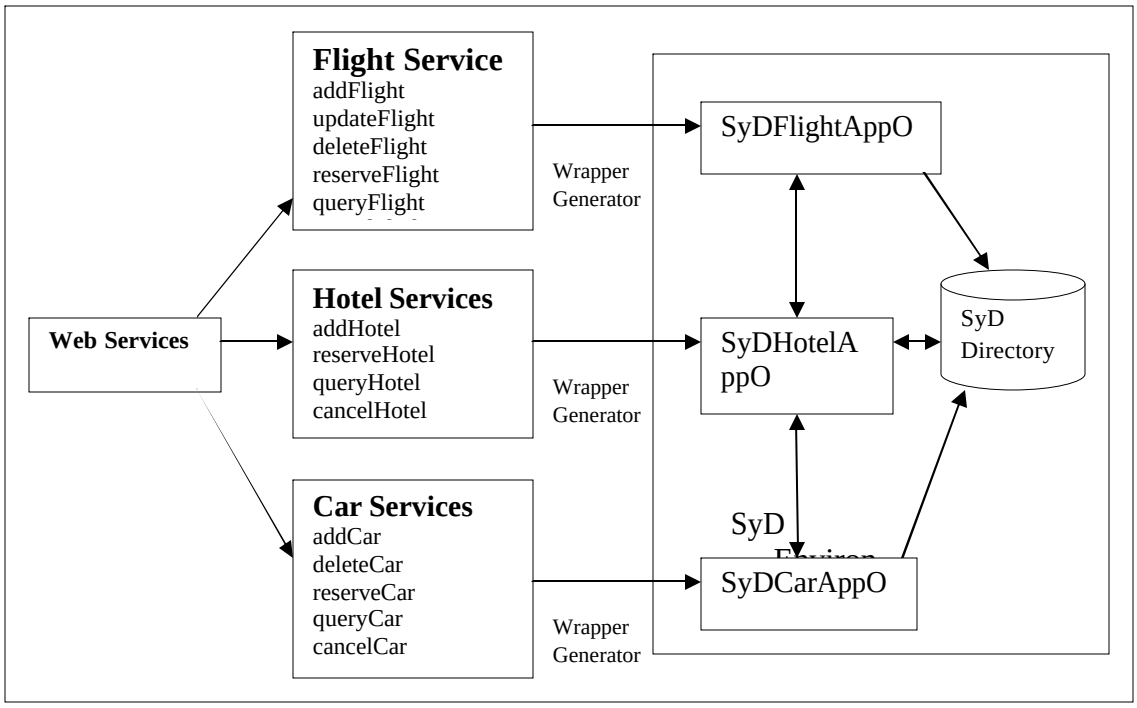

Figure 15 Travel application web services

For making the wrappers SyD enables, these wrapper objects are converted into SyD applications (WrapperAppO) corresponding to flight, hotel and car services. The methods bearing bonds are invoked and the bond execution is observed.

Subscription Bond Scenarios	**No of bonds**	**Subscription Bonds (sourceMethod --> destMethod means *Subscription* bond from srcMethod to destMethod)**
Scenario1	2	CancelFlight-->CancelHotel, CancelFlight-->CancelCar
Scenario2	3	ReserveFlight-> ReserveHotel ReserveFlight->AddHotel ReserveHotel->ReserveCar
Scenario 3	4	UpdateFlight->queryHotel QueryHotel->updateHotel UpdateHotel->queryCar QueryCar->updateCar

Figure 16 Subscription Bond Scenarios[1]

6.2 More Example Scenarios for SyD Wrapper Framework

Booking a trip through a travel agency can be one good example where web bonds can help practically. Assume three entities in this scenario, viz., user, Travel agent, Travel company. Last two entities can carry an iPAQs with SyD Wrapper application objects installed with relevant bonds installed on each. The travel agent can be a mobile entity. There can be a subscription bond between bookTicket operation at the travel agent's side and bookTicket operation at the travel company's end.

[1] The example scenarios are imaginary and may deviate from actual scenarios

When a user contacts the travel agent for booking the ticket, the travel agent, upon successful transaction at the user end, immediately invokes the bookTicket operation of the actual travel company and the transaction is complete.

The second practical scenario involves a producer and a consumer. Place order operation at the consumer side immediately invokes analyze order and subsequently accept or deny order and the producer's end if those services and methods are bonded by subscription bond using SyD Wrapper Framework and the operation can be carried out even if the entities are mobile.

6.3 Performance Analysis

The travel application experiment is tested for various kinds of coordination scenarios. Appendix A presents the results of all the tests in tabular format.

Table 1 compares number of methods, size of web service wrapper and time required for wrapper creation.

Web Service Description	Number of methods	Wrapper Creation Time (ms)	Wrapper Size (K)
Flight	6	113	12
Hotel	4	86	11
Car	5	97	12
Weather-Temperature*	1	33	4
eBay Price Watcher*	1	27	4
California Traffic Condition*	1	25	4
Barnes & Noble Price *	1	29	4

* xmethods.com web services

Table 1 SyD Wrappers Timings

From the table, it is clear that the size of the wrapper (12K) and wrapper creation time (113ms) is quite small for an average sized web service (6 methods). As the number of methods increases, size of wrapper and creation time gradually increases.

Table 2 and 3 compare number of bonds and time taken for bond creation and deletion for both subscription and negotiation bonds. We can see that the timings are nearly the same for both the tables, as creation and deletion methods don't differ much in both the cases. Bond creation time is observed to be slightly lesser than bond deletion time. This is due to internal implementation of nanoXML API for deleting a node from an XML document. Note that, when the bond is deleted, its entry is removed from the persistent XML storage.

Number of Bonds	**Bond Creation Time (ms)**	**Bond Deletion Time (ms)**
1	70	107
2	131	198
4	258	376
6	350	511

Table 2 Subscription Bond Timings

Number Of Bonds	**Bond Creation Time (ms)**	**Bond Deletion Time (ms)**
1	76	105
2	137	180
4	269	372
6	380	530

Table 3 Negotiation Bonds Timings

Tables 4 and 5 present the timings for Wrapper-SyD integrated scenario with respect to the travel application. Table 4 gives the overview of registration time for a wrapper with

the SyDDirectory. Bigger the web service in terms of number of methods, greater is the time for Wrapper registration. But the registration time is still less than a second.

Web Service Name	**No of methods**	**Wrapper Registration Time (ms)**
Flight	6	669
Hotel	4	510
Car	5	553

Table 4 Wrapper Registration with SyDDirectory

We have already discussed the travel application scenario. Table 5 discusses the distributed bond execution time for travel application with respect to the number of bonds. In this distributed approach. Each service communicates with the SyD directory for getting the description of other services. So if there is a bond between two entities.

Number Of Bonds (Subscription)	**Description (sourceMethod --> destMethod means** *Subscription* **bond from srcMethod to destMethod)**	**Bond Execution Time (ms)**
1	CancelFlight-->CancelHotel	432
2	CancelFlight-->CancelHotel, CancelFlight-->CancelCar	692
3	ReserveFlight-> ReserveHotel ReserveFlight->AddHotel ReserveHotel->ReserveCar	1392
4	UpdateFlight->queryHotel QueryHotel->updateHotel UpdateHotel->queryCar QueryCar->updateCar	2031

Table 5 Distributed Travel Application Execution Time with SyD

The source entity has to get the information of destination from SyDDirectory. That adds little bit of delay in the entire bond execution operation but nonetheless makes it effective. For executing a single bond, half a second of time is sufficient. As the number of bonds to be executed increases, the time taken for execution also increases gradually. The underlying communication timings of SyD are also considered while measuring the bond execution time.

Chapter 7 Related Work

[36] proposes using Petri nets for web service composition. Petri nets (Petri 1962, Peterson 1981) are a well founded process modeling technique that has formal semantics. They have been used to model and analyze several types of processes including protocols, manufacturing systems, and business processes [36]. They express formal semantics of the composition operators in terms of Petri nets by providing direct mapping from each Petri net to each operator. They use algebra properties to transform, optimize and compose the web service.

BPEL4WS is highly talked standard and is said to combine the best standards for Web services composition, such as IBM's WSFL and Microsoft's XLANG. XLANG is a block-structured language with basic control flow structures such as sequence, switch, while, all (for parallel routing), and pick (for race conditions based on timing or external triggers) [24]. In contrast to XLANG, WSFL is not limited to block structures and allows for directed graphs [14]. BPEL4WS allows for a mixture of block- and graph structured process models, thus making the language expressive at the price of being complex [26]. 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 [15]. It defines a protocol for interaction among web services in order to accomplish an application task. This standard contains a series of specifications from an industry group that includes IBM, Microsoft, and BEA Systems.

These abundant number of languages/standards still failed to give a framework which was fundamentally sound and yet powerful in operation. To overcome this problem, initially, a critical evaluation of these standards is required.

The approach taken by service composition and coordination standards is divided among static, dynamic or automatic and semi automatic operation. Second category is centralized Vs distributed composition and execution. Thirdly, support for wired or wireless infrastructure.

Some of current web service composition and coordination architectures inherently assume that services are resident on the wired infrastructure. Some of the examples are eFlow[28], CMI[30], Ninja service Composition architecture [29]. They assume the services to be connected to each other through stable wired network and reside on nodes or devices that are connected to each other over high bandwidth communication channels. Many of such researchers have proposed architectures that are centralized and consist of preconfigured settings for composing as well as coordinating the web services. The tasks of such pre-configuration manager are service discovery, creation of centralized coordination logic, appropriate combination of different services and management, creation of service paths, coordination of components and managing information flow among composed web services. Some of the notable limitations of centralized approach are,

(1) Central point of failure: Centralized design approach of wired infrastructure based composition architecture is highly susceptible for central point of failure, clogging of network and resources.

(2). Mobility: Some of current composition architectures do not support mobility and so large family of mobile devices deprive from composite collaborative application that use web services.

(3) Fault tolerance: Centralized composition architectures need to be adaptive to typical failures that a mobile network experiences, such as disconnection, node failure etc.

[6] describes the 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 [6]. The composition protocols are based on distributed brokerage mechanisms and utilize a distributed service discovery process over ad-hoc network connectivity.

Chapter 8 Conclusion and future work

8.1 Conclusion

We have proposed and presented SyD Wrapper approach in the field of web service composition and coordination. Web service coordination is achieved by using web coordination bonds [11] in the SyD Wrappers. These wrappers hide the bond creation and implementation logic from the application developer as well as user. They encapsulate bond coordination logic along with calls to original web service. For a developer, there is no difference between calling the wrapper object and calling the original web service method.

Biggest advantage of the wrapper approach is that it can be seamlessly fit in small mobile devices running on SyD platform. Collaborative application development for small handheld devices can be carried out with comparatively less complexity using web service wrappers.

To support the claim, SyD wrappers are integrated with small devices running SyD middleware and a travel application scenario is created. Bonds are created among various methods of the travel web services (flight, hotel and car). Bond execution is showcased and performance of this approach is also analyzed.

8.2 Future Work

In this thesis, we proposed and implemented the novel approach of SyD Wrappers for composition and coordination of web service. Web coordination bonds [11] form the basis of the coordination logic.

The Wrapper Framework works well in the SyD Environment. Experiments with SyD Wrapper on iPAQs running SyD middleware is the priority work.

To showcase the bond coordination power, we have implemented subscription bonds that deal with automatic flow of information among bonded entities as a part of coordination bond scenario. Subscription bonds give the fair picture of how wrappers behave when the bond among different entities is executed. Complete implementation of the negotiation bond, which deals with constraint enforcement, is the next important step to be carried out in future. Negotiation bonds will not only give more power to this architecture, but also reveal the wrapper behavior in some complex situations.

Implementation of tentative bonds which allows an interested party to wait for bond creation (preferably in a queue) until the desired entity become eligible for bonding is also one of the important task on the list.

Comparison of the web service wrapper approach in terms of performance evaluation with similar technologies is next important work to be carried out in future.

Bibliography

[1] IBM Web Services Architecture Team, "Web Services Architecture Overview," *http://www-106.ibm.com/developerworks/webservices/library/w-ovr/*, September, 2002.

[2] World Wide Web Consortium, "Web Services – Design Issues," *http://www.w3.org/-DesignIssues/*

[3] World Wide Web Consortium, "WSDL Web-service Description Language," *http://www.w3.org/TR/wsdl* , 2001

[4] "Universal Description, Discovery and Integration of Web Services" (UDDI) 3, *http://www.oasis-open.org/committees/uddi-spec/tcspecs.shtml#uddiv3, 2002*

[5] 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.

[6] 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

[7] 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. 9-18.

[8] 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

[9] Johnson P Thomas, Mathews Thomas, George Ghinea, “Modeling of Web Services Flow,” *IEEE Intl. Conf. on ECommerce,* June 24 - 27, 2003 Newport Beach, California , pp. 331-339.

[10] 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

[11] 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

[12] 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.

[13]W.M.P. van der Aalst, Workflow patterns, *http:/ tmitwww.tm.tue.nl/research/patterns,* 2003.

[14] F. Leymann, “Web Services Flow Language (WSFL 1.0),” IBM, May 2001, *http://www-3.ibm.com/software/solutions /webservices/pdf/WSFL.pdf*

[15] Web Services Coordination, http://www-106.ibm.com/developerworks/library/ws-coor/

[16] 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

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

[18] Sushil K. Prasad, Anu G. Bourgeois, Erdogan Dogdu, 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.

[19] Staab et al, “Web Services: Been there done that”, Intelligent Systems, IEEE, Volume: 18, Issue: 1 , Jan.-Feb. 2003 , pp 72 – 85

[20] Skogan, D.; Gronmo, R.; Solheim, I.; Enterprise Distributed Object Computing Conference, 2004. EDOC 2004. Proceedings. Eighth IEEE International , 20-24 Sept. 2004, pp 47 - 57

[21] Vijay Madisetti, “SyD: A middleware infrastructure for mobile iAppliance devices," *EE Times Network, http://www.iapplianceWeb.com/story/OEG20021105S0031,* November 5, 2002.

[22] S. Thöne, R. Depke, and G. Engels, “Process-Oriented, Flexible Composition of Web Services with UML,” *Proc. of the Int. Workshop on Conceptual Modeling Approaches for e-Business: A Web Service Perspective (eCOMO 2002),* Tampere, Finland, Oct. 2002, Springer LNCS 2784,

[23] F. Curbera, et al., "Business Process Execution Language for Web Services (Version 1.0)," IBM, July 2002, http*:// www-106.ibm.com/developerworks/webservices/library/ws-bpel.*

[24] S. Thatte. XLANG: Web Services for Business Process Design. Microsoft, 2001

[25] F. Leymann. Web Services Flow Language (WSFL 1.0). IBM, May 2001.

[26] 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

[27] W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Web Service Composition Languages: Old Wine in New Bottles? In G. Chroust and C. Hofer, editors, *Proc. of the 29th EUROMICRO Conf. on New Waves in System Architecture,* Los Alamitos, CA, 2003, pp. 8-305.

[28] Fabio Casati, Ski Ilnicki, LiJie Jin, Vasudev Krishnamoorthy, Ming-Chien Shan, Adaptive and Dynamic Service Composition in eFlow, *HP Technical Report*, HPL-2000-39, March, 2000

[29] 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.

[30] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. *Modeling and composing service-based and reference process-based multi-enterprise processes*. In Proc. of the Int. Conference on Advanced Information Systems, 2000

[31] NanoXML parser, http://nanoxml.cyberelf.be/downloads/NanoXML-Java.pdf

[32]IBM, WSDL4J project,

http://www-124.ibm.com/developerworks/projects/wsdl4j/

[33] SOAP, http://www.perfectxml.com/soap.asp

[34] Axis 1.1 Documentation, http://ws.apache.org/axis

[35] R. Hamadi, B. Benatalah, *A Petri net-based model for web service composition*, Conferences in Research and Practice in Information Technology Series, 2003

[36] Arthi Hariharan, Anu Bourgeois, *Constraint Based Collaborative Web Services: A Framework*, Masters Thesis, 2003

Appendix A – Wrapper API

Packages	
UI	
Wrapper	
WSMS	

UI

Class Menu

java.lang.Object
 |
 +--**UI.Menu**

public class **Menu**
extends java.lang.Object

Entry point to the SyD Wrapper Generator System

Constructor Summary	
Menu()	

Method Summary	
static void	**main**(java.lang.String[] args) The main program for the Menu class
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Constructor Detail

Menu

public **Menu**()

Method Detail

main

public static void **main**(java.lang.String[] args)
The main program for the Menu class

Parameters:
args - The command line arguments This class is entry point to the system It shows list of operations

---

WSMS

CLASS PARSEWSDL
java.lang.Object
|
+--**WSMS.ParseWsdl**

---

public class **ParseWsdl**
extends java.lang.Object

Description of the Class

Constructor Summary

ParseWsdl()

Method Summary

java.util.Hashtable	**parseWsdl**(java.lang.String wsdlUrl) This method parses the wsdl file of a web service and returns method and paramter list
Methods inherited from class java.lang.Object	
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait	

Constructor Detail

ParseWsdl

public **ParseWsdl**()

Method Detail

parseWsdl

public java.util.Hashtable **parseWsdl**(java.lang.String wsdlUrl)
throws javax.wsdl.WSDLException

This method parses the wsdl file of a web service and returns method and paramter list

Parameters:

wsdlUrl - wsdl url of the web service

Returns:

Hashtable containing methods and parameters

Throws:

javax.wsdl.WSDLException - WSDL parsing Exception

WSMS

Class SearchWS

java.lang.Object
|
+--**WSMS.SearchWS**

---

public class **SearchWS**
extends java.lang.Object

Description of the Class

Constructor Summary

SearchWS()	

Method Summary

void	**search**() This method accepts wsdl url as an input parameter from the user and invokes parseWsdl method It prompts user for saving the web service and passes control to storeWS and Wrapper classes accordingly

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

SearchWS

public **SearchWS**()

Method Detail

search

public void **search**()

This method accepts wsdl url as an input parameter from the user and invokes parseWsdl method It prompts user for saving the web service and passes control to storeWS and Wrapper classes accordingly

---

WSMS

Class StoreWS

java.lang.Object
 |
 +--**WSMS.StoreWS**

---

public class **StoreWS**
extends java.lang.Object

Description of the Class

Constructor Summary

StoreWS()	

Method Summary

void	**createFile**(java.lang.String fileName) Creates the XML data file
java.lang.String	**getWSName**(java.lang.String wsdlUrl) Gets the Web Service name from the wsdl url
void	**store**(java.lang.String wsdlUrl, java.util.Hashtable methodHash) This method stores the WSDL details in an XML file

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

StoreWS
public **StoreWS**()

Method Detail

store
public void **store**(java.lang.String wsdlUrl,
java.util.Hashtable methodHash)
This method stores the WSDL details in an XML file
Parameters:
wsdlUrl - url of the wsdl
methodHash - Hashtable containing method and parameter details

---

getWSName
public java.lang.String **getWSName**(java.lang.String wsdlUrl)
Gets the Web Service name from the wsdl url
Parameters:
wsdlUrl - Description of the Parameter
Returns:
The wSName value

---

createFile
public void **createFile**(java.lang.String fileName)
throws java.io.IOException
Creates the XML data file
Parameters:
fileName - Description of the Parameter
Throws:
java.io.IOException - Description of the Exception

WSMS

Class ViewWS

java.lang.Object
|
+--**WSMS.ViewWS**

---

public class **ViewWS**
extends java.lang.Object

This class facilitates the user to view the details for stores Web Service

---

Constructor Summary

ViewWS()	

Method Summary

void	**view**() Details of the stored Web Service in terms of method names, parameter names and data types can be viewed using this method This method uses nanoXML API for sml parsing

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ViewWS

public **ViewWS**()

Method Detail

view

public void **view**()
throws java.lang.Exception

Details of the stored Web Service in terms of method names, parameter names and data types can be viewed using this method This method uses nanoXML API for sml parsing

Throws:

java.lang.Exception - XML parsing exception

Wrapper

Class Wrapper

java.lang.Object
|
+--**Wrapper.Wrapper**

---

public class **Wrapper**
extends java.lang.Object
This class deals with all the operations on the wrapper inclusing creating, executing, checking a wrapper For every web service, two files are created, viz, __wsName__.java and __wsName__Impl.java. Both these files are SyD enabled. __wsName__.java file is an interface while __wsName__Impl.java is its implementation Both the files are RMI enabled, which is a specification for SyD for SySD directory and remote object invocation

---

Constructor Summary

Wrapper()	

Method Summary

void	**createWrapper**(java.lang.String wsdlUrl, java.util.Hashtable methodHash) This method creates the wrapper for a web service
void	**writeCheckForBonds**() This method writes the writeCheckForBonds method to the wrapper file
void	**writeExecuteSubscriptionBond**() This method writes the executeSubscriptionBond method to the wrapper file

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

Wrapper

public **Wrapper**()

Method Detail

createWrapper
public void **createWrapper**(java.lang.String wsdlUrl,
java.util.Hashtable methodHash)
This method creates the wrapper for a web service
Parameters:
wsdlUrl - url for the wsdl file
methodHash - hashtable containing method and parameter list

---

writeExecuteSubscriptionBond
public void **writeExecuteSubscriptionBond**()
This method writes the executeSubscriptionBond method to the wrapper file

---

writeCheckForBonds
public void **writeCheckForBonds**()
This method writes the writeCheckForBonds method to the wrapper file

---

Wrapper
Class DeleteBond
java.lang.Object
|
+--**Wrapper.DeleteBond**

---

public class **DeleteBond**
extends java.lang.Object
Bond deletion is achieved through this method

Constructor Summary

DeleteBond()

Method Summary

void	**deleteWebBond**() This method details of a bond from XML storage

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

DeleteBond
public **DeleteBond**()

Method Detail

deleteWebBond
public void **deleteWebBond**()
This method details of a bond from XML storage

Wrapper

Class ViewBond

java.lang.Object
 |
 +--**Wrapper.ViewBond**

---

public class **ViewBond**
extends java.lang.Object
This method facilitates the user to view the bond information

---

Constructor Summary

ViewBond()	

Method Summary

void	**viewWebBond**() Bond information is shown by this method in the form of source and destination web service as well as method names, type of bond etc

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

ViewBond
public **ViewBond**()

Method Detail

viewWebBond
public void **viewWebBond**()
Bond information is shown by this method in the form of source and destination web service as well as method names, type of bond etc

---

Example Wrapper Interface

syd.sydapp
Interface flight
All Superinterfaces:
java.rmi.Remote
All Known Implementing Classes:
flightImpl

---

public interface **flight**
extends java.rmi.Remote
This is the interface for flight SyD Wrapper Object The actual implementation class is flightImpl

---

Method Summary

java.lang.String	**addFlight**(java.lang.Integer in0, java.lang.String in1, java.lang.String in2, java.lang.String in3)
java.lang.String	**cancelFlight**(java.lang.String in0, java.lang.String in1)
java.lang.String	**deleteFlight**(java.lang.String in0)
java.lang.String	**queryFlight**(java.lang.String in0, java.lang.String in1)
java.lang.String	**reserveFlight**(java.lang.String in0, java.lang.String in1)
java.lang.String	**updateFlight**(java.lang.Integer in0, java.lang.String in1, java.lang.String in2, java.lang.String in3)

Method Detail

addFlight
public java.lang.String **addFlight**(java.lang.Integer in0,
java.lang.String in1,
java.lang.String in2,
java.lang.String in3)
throws java.rmi.RemoteException
java.rmi.RemoteException

---

cancelFlight
public java.lang.String **cancelFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException
java.rmi.RemoteException

---

reserveFlight
public java.lang.String **reserveFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException
java.rmi.RemoteException

---

deleteFlight
public java.lang.String **deleteFlight**(java.lang.String in0)
throws java.rmi.RemoteException
java.rmi.RemoteException

---

queryFlight
public java.lang.String **queryFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException
java.rmi.RemoteException

---

updateFlight
public java.lang.String **updateFlight**(java.lang.Integer in0,
java.lang.String in1,
java.lang.String in2,
java.lang.String in3)
throws java.rmi.RemoteException
java.rmi.RemoteException

Example Wrapper Class

syd.sydapp

Class flightImpl

```
java.lang.Object
  |
  +--java.rmi.server.RemoteObject
        |
        +--java.rmi.server.RemoteServer
              |
              +--java.rmi.server.UnicastRemoteObject
                    |
                    +--syd.sydapp.flightImpl
```

All Implemented Interfaces:

flight, java.rmi.Remote, java.io.Serializable

---

public class **flightImpl**
extends java.rmi.server.UnicastRemoteObject
implements flight

This is the SyD wrapper class for flight web service It extends the flight interface

See Also:

Serialized Form

Field Summary

Fields inherited from class java.rmi.server.RemoteObject
Ref

Constructor Summary

flightImpl() Constructor for the flightImpl object

Method Summary

java.lang.String	**addFlight**(java.lang.Integer in0, java.lang.String in1, java.lang.String in2, java.lang.String in3) This method is a wrapper method for the actual addFlight method of the original webs service.
java.lang.String	**cancelFlight**(java.lang.String in0, java.lang.String in1)
Boolean	**checkForBonds**(java.lang.String methodName, java.lang.String wrapperFile,

	java.lang.String bondType) This method checks for the existance of bond for a particular method It parses the wrapper file and checks if it contains the specified bond (subscription/negotiation)
java.lang.String	**deleteFlight**(java.lang.String in0)
void	**executeSubscriptionBond**(java.lang.String methodName, java.lang.String wFile) This method executes the subscription bond by invoking the the destination method of a web service with which the source method has a bond
java.lang.String	**queryFlight**(java.lang.String in0, java.lang.String in1)
java.lang.String	**reserveFlight**(java.lang.String in0, java.lang.String in1)
java.lang.String	**updateFlight**(java.lang.Integer in0, java.lang.String in1, java.lang.String in2, java.lang.String in3)

Methods inherited from class java.rmi.server.UnicastRemoteObject
clone, exportObject, exportObject, exportObject, unexportObject

Methods inherited from class java.rmi.server.RemoteServer
getClientHost, getLog, setLog

Methods inherited from class java.rmi.server.RemoteObject
equals, getRef, hashCode, toString, toStub

Methods inherited from class java.lang.Object
finalize, getClass, notify, notifyAll, wait, wait, wait

Constructor Detail

flightImpl
public **flightImpl**()
throws java.rmi.RemoteException
Constructor for the flightImpl object

Method Detail

addFlight

public java.lang.String **addFlight**(java.lang.Integer in0,
java.lang.String in1,
java.lang.String in2,
java.lang.String in3)
throws java.rmi.RemoteException

This method is a wrapper method for the actual addFlight method of the original webs service. It calls the method to check for existance of bond and execute the bonds if any. It eventually makes a SOAP call to the original web service All the subsequent wrapper methods have the same functionality

Specified by:
addFlight in interface flight
java.rmi.RemoteException

---

cancelFlight

public java.lang.String **cancelFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException

Specified by:
cancelFlight in interface flight
java.rmi.RemoteException

---

reserveFlight

public java.lang.String **reserveFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException

Specified by:
reserveFlight in interface flight
java.rmi.RemoteException

---

deleteFlight

public java.lang.String **deleteFlight**(java.lang.String in0)
throws java.rmi.RemoteException

Specified by:
deleteFlight in interface flight
java.rmi.RemoteException

---

queryFlight

public java.lang.String **queryFlight**(java.lang.String in0,
java.lang.String in1)
throws java.rmi.RemoteException

Specified by:
queryFlight in interface flight
java.rmi.RemoteException

---

updateFlight

public java.lang.String **updateFlight**(java.lang.Integer in0,
java.lang.String in1,
java.lang.String in2,
java.lang.String in3)
throws java.rmi.RemoteException

Specified by:
updateFlight in interface flight
java.rmi.RemoteException

---

checkForBonds
public boolean **checkForBonds**(java.lang.String methodName,
java.lang.String wrapperFile,
java.lang.String bondType)
This method checks for the existance of bond for a particular method It parses the wrapper file and checks if it contains the specified bond (subscription/negotiation)

Parameters:
methodName - method name
wrapperFile - name of the wrapper file
bondType - type of the bond

Returns:
boolean value to indicate presence of bond

---

executeSubscriptionBond
public void **executeSubscriptionBond**(java.lang.String methodName,
java.lang.String wFile)
throws java.lang.Exception
This method executes the subscription bond by invoking the the destination method of a web service with which the source method has a bond

Parameters:
methodName - source method name
wFile - wrapper file name
java.lang.Exception

Appendix B - Source Code

```
package syd.sydapp;

import java.rmi.Remote;
import java.rmi.RemoteException;

/**
 *  This is the interface for flight SyD Wrapper Object
 *  The actual implementation class is flightImpl
 * @author     Mohini Padhye
 * @created    November 10, 2004
 */
public interface flight extends Remote {
 public String addFlight(Integer in0, String in1, String in2,
String in3) throws RemoteException;
 public String cancelFlight(String in0, String in1) throws
RemoteException;
 public String reserveFlight(String in0, String in1) throws
RemoteException;
 public String deleteFlight(String in0) throws RemoteException;
 public String queryFlight(String in0, String in1) throws
RemoteException;
 public String updateFlight(Integer in0, String in1, String in2,
String in3) throws RemoteException;
}
```

```
package syd.sydapp;

import java.io.*;
import java.net.*;
import java.util.*;
import java.lang.reflect.*;
import net.n3.nanoxml.*;

import org.apache.axis.client.*;
import org.apache.axis.encoding.*;

import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

import org.apache.axis.utils.*;
import javax.xml.namespace.QName;
import javax.xml.rpc.ParameterMode;

/**
 *  This is the SyD wrapper class for flight web service
```

```
 *  It extends the flight interface
 * @author     Mohini Padhye
 * @created    November 23, 2004
 */
public class flightImpl extends UnicastRemoteObject implements
flight {
 String baseDir = "syd/sydapp";

 /**
  *Constructor for the flightImpl object
  */
 public flightImpl() throws RemoteException {
  super();
 }

 /**
  *  This method is a wrapper method for the actual addFlight
method of the
  *  original webs service. It calls the method to check for
existance of
  *  bond and execute the bonds if any. It eventually makes a
SOAP call to
  *  the original web service
  *  All the subsequent wrapper methods have the same
functionality
  */
 public String addFlight(Integer in0, String in1, String in2,
String in3) throws RemoteException {
  String methodName = "addFlight";
  String wrapperFile = baseDir + "/flight.xml";
  String retVal = " ";

  try {
   File wrapperF = new File(wrapperFile);
   Service service = new Service();
   Call call = (Call) service.createCall();
   call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
   call.setOperationName(new QName("urn:flight", "addFlight"));
   call.addParameter("in0", XMLType.XSD_INT, ParameterMode.IN);
   call.addParameter("in1", XMLType.XSD_STRING,
ParameterMode.IN);
   call.addParameter("in2", XMLType.XSD_STRING,
ParameterMode.IN);
   call.addParameter("in3", XMLType.XSD_STRING,
ParameterMode.IN);
   call.setReturnType(XMLType.XSD_STRING);
   /* Invoking actual web service method */
```

```
    retVal = (String) call.invoke(new Object[]{in0, in1, in2,
in3});
    /* Checking for subscription bonds */
    if (wrapperF.exists()) {
     boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
     if (isSubBond) {
      /*  Executing subscription bond */
      this.executeSubscriptionBond(methodName, wrapperFile);
     }
    }
   } catch (Exception e) {
    System.out.println(e.toString());
   }
   return retVal;
  }

  public String cancelFlight(String in0, String in1) throws
RemoteException {
   String methodName = "cancelFlight";
   String wrapperFile = baseDir + "/flight.xml";
   String retVal = " ";

   try {
    File wrapperF = new File(wrapperFile);
    if (wrapperF.exists()) {
     boolean isNegBond = this.checkForBonds(methodName,
wrapperFile, "N");
    }
    Service service = new Service();
    Call call = (Call) service.createCall();
    call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
    call.setOperationName(new QName("urn:flight",
"cancelFlight"));
    call.addParameter("in0", XMLType.XSD_STRING,
ParameterMode.IN);
    call.addParameter("in1", XMLType.XSD_STRING,
ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);
    retVal = (String) call.invoke(new Object[]{in0, in1});
    System.out.println("Return Value:" + retVal);
    if (wrapperF.exists()) {
     boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
     if (isSubBond) {
      this.executeSubscriptionBond(methodName, wrapperFile);
     }
    }
```

```
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return retVal;
 }

 public String reserveFlight(String in0, String in1) throws
RemoteException {
  String methodName = "reserveFlight";
  String wrapperFile = baseDir + "/flight.xml";
  String retVal = " ";

  try {
   File wrapperF = new File(wrapperFile);
   if (wrapperF.exists()) {
    boolean isNegBond = this.checkForBonds(methodName,
wrapperFile, "N");
   }
   Service service = new Service();
   Call call = (Call) service.createCall();
   call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
   call.setOperationName(new QName("urn:flight",
"reserveFlight"));
   call.addParameter("in0", XMLType.XSD_STRING,
ParameterMode.IN);
   call.addParameter("in1", XMLType.XSD_STRING,
ParameterMode.IN);
   call.setReturnType(XMLType.XSD_STRING);
   retVal = (String) call.invoke(new Object[]{in0, in1});
   System.out.println("Return Value:" + retVal);
   if (wrapperF.exists()) {
    boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
    if (isSubBond) {
     this.executeSubscriptionBond(methodName, wrapperFile);
    }
   }
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return retVal;
 }

 public String deleteFlight(String in0) throws RemoteException {
  String methodName = "deleteFlight";
  String wrapperFile = baseDir + "/flight.xml";
  String retVal = " ";
```

```
  try {
   File wrapperF = new File(wrapperFile);
   if (wrapperF.exists()) {
    boolean isNegBond = this.checkForBonds(methodName,
wrapperFile, "N");
   }
   Service service = new Service();
   Call call = (Call) service.createCall();
   call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
   call.setOperationName(new QName("urn:flight",
"deleteFlight"));
   call.addParameter("in0", XMLType.XSD_STRING,
ParameterMode.IN);
   call.setReturnType(XMLType.XSD_STRING);
   retVal = (String) call.invoke(new Object[]{in0});
   System.out.println("Return Value:" + retVal);
   if (wrapperF.exists()) {
    boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
    if (isSubBond) {
     this.executeSubscriptionBond(methodName, wrapperFile);
    }
   }
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return retVal;
 }

 public String queryFlight(String in0, String in1) throws
RemoteException {
  String methodName = "queryFlight";
  String wrapperFile = baseDir + "/flight.xml";
  String retVal = " ";

  try {
   File wrapperF = new File(wrapperFile);
   if (wrapperF.exists()) {
    boolean isNegBond = this.checkForBonds(methodName,
wrapperFile, "N");
   }
   Service service = new Service();
   Call call = (Call) service.createCall();
   call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
   call.setOperationName(new QName("urn:flight", "queryFlight"));
   call.addParameter("in0", XMLType.XSD_STRING,
ParameterMode.IN);
```

```
    call.addParameter("in1", XMLType.XSD_STRING,
ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);
    retVal = (String) call.invoke(new Object[]{in0, in1});
    System.out.println("Return Value:" + retVal);
    if (wrapperF.exists()) {
     boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
     if (isSubBond) {
      this.executeSubscriptionBond(methodName, wrapperFile);
     }
    }
   } catch (Exception e) {
    System.out.println(e.toString());
   }
   return retVal;
  }

  public String updateFlight(Integer in0, String in1, String in2,
String in3) throws RemoteException {
   String methodName = "updateFlight";
   String wrapperFile = baseDir + "/flight.xml";
   String retVal = " ";

   try {
    File wrapperF = new File(wrapperFile);
    if (wrapperF.exists()) {
     boolean isNegBond = this.checkForBonds(methodName,
wrapperFile, "N");
    }
    Service service = new Service();
    Call call = (Call) service.createCall();
    call.setTargetEndpointAddress(new
java.net.URL("http://localhost:8080/axis/services/flight"));
    call.setOperationName(new QName("urn:flight",
"updateFlight"));
    call.addParameter("in0", XMLType.XSD_INT, ParameterMode.IN);
    call.addParameter("in1", XMLType.XSD_STRING,
ParameterMode.IN);
    call.addParameter("in2", XMLType.XSD_STRING,
ParameterMode.IN);
    call.addParameter("in3", XMLType.XSD_STRING,
ParameterMode.IN);
    call.setReturnType(XMLType.XSD_STRING);
    retVal = (String) call.invoke(new Object[]{in0, in1, in2,
in3});
    System.out.println("Return Value:" + retVal);
    if (wrapperF.exists()) {
```

```
    boolean isSubBond = this.checkForBonds(methodName,
wrapperFile, "S");
    if (isSubBond) {
     this.executeSubscriptionBond(methodName, wrapperFile);
    }
   }
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return retVal;
 }



 /**
  *  This method checks for the existance of bond for a
particular method
  *  It parses the wrapper file and checks if it contains the
specified
  *  bond (subscription/negotiation)
  * @param  methodName   method name
  * @param  wrapperFile  name of the wrapper file
  * @param  bondType     type of the bond
  * @return             boolean value to indicate presence of
bond
  */
 public boolean checkForBonds(String methodName, String
wrapperFile, String bondType) {
  String bType;
  try {
   File wrapperF = new File(wrapperFile);
   if (!wrapperF.exists()) {
    System.out.println("XML data file does not exist");
    return false;
   }
   IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
   IXMLReader reader = StdXMLReader.fileReader(wrapperFile);
   parser.setReader(reader);
   IXMLElement mainRoot = (IXMLElement) parser.parse();
   if (mainRoot.getChildrenCount() > 1) {
    IXMLElement root = (IXMLElement) mainRoot.getChildAtIndex(1);
    for (int i = 0; i < root.getChildrenCount(); i++) {
     IXMLElement bond = (IXMLElement) root.getChildAtIndex(i);
     for (int j = 0; j < bond.getChildrenCount(); j++) {
      bType = ((IXMLElement)
bond.getChildAtIndex(3)).getContent();
      if (bType.equals(bondType)) {
       return true;
      }
```

```
      }
     }
    }
   } catch (Exception e) {
    e.printStackTrace();
   }
   return false;
  }

  /**
   *  This method executes the subscription bond by invoking the
   *  the destination method of a web service with which
   *  the source method has a bond
   *
   * @param  methodName     source method name
   * @param  wFile          wrapper file name
   */
  public void executeSubscriptionBond(String methodName, String
wFile) throws Exception {
   try {
    boolean hasBond = false;
    String destWS = "";
    String destMethod = "";
    String srcMethod;
    Vector paramVector = new Vector();
    Vector paramType = new Vector();
    IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
    IXMLReader reader = StdXMLReader.fileReader(wFile);
    parser.setReader(reader);
    IXMLElement fileRoot = (IXMLElement) parser.parse();
    IXMLElement root = (IXMLElement) fileRoot.getChildAtIndex(1);
    for (int i = 0; i < root.getChildrenCount(); i++) {
     IXMLElement bond = (IXMLElement) root.getChildAtIndex(i);
     for (int j = 0; j < bond.getChildrenCount(); j++) {
      /* Retrieving destination method and service name from XML
storage */
      if ((((IXMLElement)

bond.getChildAtIndex(j++)).getContent()).equals(methodName)) {
       destWS = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent();
       destMethod = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent();
       String bondType = ((IXMLElement)
        bond.getChildAtIndex(j++)).getContent();
       String trigger = ((IXMLElement)
        bond.getChildAtIndex(j++)).getContent();
       if (bondType.equals("S") && trigger.equals("Y")) {
        hasBond = true;
```

```
        /* Invoking the SyD WrapperAppO */
        WrapperAppO wAppO = new WrapperAppO("S");
        wAppO.setVisible(true);
        wAppO.showDestParamInputFrame(destWS,destMethod);
        paramType = wAppO.getDestParamType();
        paramVector=wAppO.getDestParam();
        /* Getting list of destination parameters from the SyD
WrapperAppO*/
        paramVector = wAppO.getDestParams();
       }
      }
     }
     if (hasBond == false) {
      return;
     } else {
      /*Invoking the actual destination method*/
      WrapperAppO wAppO = new WrapperAppO();
      String retVal = wAppO.invoke(destWS, destMethod, paramType,
paramVector);
      System.out.println("Return:" + retVal);
     }
     hasBond = false;
    }
   } catch (Exception e) {
    e.printStackTrace();
   }
  }
}
```

```
<!-- This file acts as a persistent storage for storing
information relating to
   the web service, it is divided in two seections, where first
section
   stores service information and second section stores bond
information -->
<WebService>
 <!-- This section contains all the necessary information about
the web
    service including name of the web service, wsdl url,
namespaceURI,
    locationURI, method names,parameter type and return type -->
 <WS>
  <WsdlUrl>flight.wsdl</WsdlUrl>
  <WSName>flight</WSName>

 <LocationURI>http://localhost:8080/axis/services/flight</Locatio
nURI>
  <NamespaceURI>urn:flight</NamespaceURI>
```

```
  <Method mid="1">
   <MethodName>reserveFlight</MethodName>
   <ParamType>int in0</ParamType>
   <ParamType>string in1</ParamType>
   <ReturnType>string reserveFlightReturn</ReturnType>
  </Method>
  <Method mid="2">
   <MethodName>updateFlight</MethodName>
   <ParamType>int in0</ParamType>
   <ParamType>string in1</ParamType>
   <ReturnType>string updateFlightReturn</ReturnType>
  </Method>
  <Method mid="3">
   <MethodName>queryFlight</MethodName>
   <ParamType>string in0</ParamType>
   <ReturnType>string queryFlightReturn</ReturnType>
  </Method>
  <Method mid="4">
   <MethodName>cancelFlight</MethodName>
   <ParamType>string in0</ParamType>
   <ParamType>string in1</ParamType>
   <ReturnType>string cancelFlightReturn</ReturnType>
  </Method>
  <Method mid="5">
   <MethodName>deleteFlight</MethodName>
   <ParamType>string in0</ParamType>
   <ReturnType>string deleteFlightReturn</ReturnType>
  </Method>
  <Method mid="6">
   <MethodName>addFlight</MethodName>
   <ParamType>int in0</ParamType>
   <ParamType>string in1</ParamType>
   <ReturnType>string addFlightReturn</ReturnType>
  </Method>
  </WS>
 <!-- This section bond related information for a web service
including source
    method, destination method, destination web service, type of
bond etc -->
 <Wrapper>
  <Bond bid="1">
   <SrcMethod>deleteFlight</SrcMethod>
   <DestWS>hotel</DestWS>
   <DestMethod>cancelHotel</DestMethod>
   <Type>S</Type>
   <Trigger>Y</Trigger>
  </Bond>
  <Bond bid="2">
   <SrcMethod>deleteFlight</SrcMethod>
```

```
   <DestWS>car</DestWS>
   <DestMethod>cancelCar</DestMethod>
   <Type>S</Type>
   <Trigger>Y</Trigger>
  </Bond>
 </Wrapper>
 </WebService>
```

```
package UI;
import java.io.*;

/**
 *  Entry point to the SyD Wrapper Generator System
 * @author     Mohini Padhye
 * @created    September 15, 2004
 */
public class Menu {
    /**
     *  The main program for the Menu class
     * @param  args  The command line arguments
     * This class is entry point to the system
     * It shows list of operations
     */
      public static void main(String[] args) {
          int choice;
          boolean bool = false;
          try {
              do {
     System.out.println("Menu\n1. Search Web Service\n2. View
Saved Web Services\n"+
      "3. Create Bond\n4. View Bonds\n5. Delete Bond\n6. View
Report\n7. Exit");
     System.out.println("Enter your choice:");
     BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
      choice = br.read();
        switch (choice) {
          case '1':
              WSMS.SearchWS searchWS = new WSMS.SearchWS();
              searchWS.search();
              break;
          case '2':
               WSMS.ViewWS viewWS = new WSMS.ViewWS();
               viewWS.view();
               break;
          case '3':
               Wrapper.CreateBond createBond = new
Wrapper.CreateBond();
```

```
                createBond.createWebBond();
                 break;
            case '4':
                Wrapper.ViewBond viewBond = new Wrapper.ViewBond();
                viewBond.viewWebBond();
                 break;
            case '5':
                Wrapper.DeleteBond deleteBond = new
Wrapper.DeleteBond();
                deleteBond.deleteWebBond();
                 break;
             case '6':
                  UI.Report report = new UI.Report();
                 report.createReport();
                  break;
            case '7':
                  bool = true;
                  break;
            }
        } while (bool == false);
        } catch (Exception e) {
         System.out.println("Exception :" + e.toString());
        }
      }
}
```

```
package WSMS;

import java.io.*;
import java.util.*;

/**
 *  Description of the Class
 *
 * @author     Mohini Padhye
 * @created    September 15, 2004
 */
public class SearchWS {
 /**
  *  This method accepts wsdl url as an input parameter
  *  from the user and invokes parseWsdl method
  *  It prompts user for saving the web service
  *  and passes control to storeWS and Wrapper classes
accordingly
  */
 public void search() {
  String wsdlUrl = "";
  String ans = "N";
```

```
  Hashtable methodHash = new Hashtable();

  try {
   BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
   System.out.println("Enter correct WSDL url :");

   wsdlUrl = br.readLine();

   WSMS.ParseWsdl pWsdl = new WSMS.ParseWsdl();
   methodHash = pWsdl.parseWsdl(wsdlUrl);

   BufferedReader br1 = new BufferedReader(new
InputStreamReader(System.in));
   System.out.println("\nDo you want to Add this web
Service?(Y/N):");
   ans = br1.readLine();

   if (ans.equals("Y") || ans.equals("y")) {
    time1 = System.currentTimeMillis ();
    WSMS.StoreWS storeWS = new WSMS.StoreWS();
    storeWS.store(wsdlUrl, methodHash);

    Wrapper.Wrapper wrapper = new Wrapper.Wrapper();
    wrapper.createWrapper(wsdlUrl, methodHash);
             System.out.println("\nWrapper Successfully
Created");
   }
  } catch (Exception e) {
   e.printStackTrace();
   System.out.println(e.toString());
  }
 }
}
```

```
package WSMS;

import javax.wsdl.*;
import com.ibm.wsdl.*;
import javax.wsdl.extensions.soap.*;
import javax.wsdl.factory.WSDLFactory;
import javax.wsdl.xml.WSDLReader;
import java.util.*;
import javax.xml.namespace.QName;

/**
 *  Description of the Class
 *
```

```
 * @author     Mohini Padhye
 * @created    September 19, 2004
 */
public class ParseWsdl {
 Hashtable wsdlHash; /* Stores method and parameter details*/
 Vector params;

 /**
  *  This method parses the wsdl file of a web service and
  *  returns method and paramter list
  * @param  wsdlUrl           wsdl url of the web service
  * @return                   Hashtable containing methods and
parameters
  * @exception  WSDLException  WSDL parsing Exception
  */

 public Hashtable parseWsdl(String wsdlUrl) throws WSDLException
{
  /* WSDL Parsing uses WSDL4J API, following methods are a part
of that API*/
  WSDLFactory wsdlfactory = WSDLFactory.newInstance();
  WSDLReader wsdlreader = wsdlfactory.newWSDLReader();
  Definition definition = wsdlreader.readWSDL(wsdlUrl);
  Iterator i = definition.getServices().values().iterator();
  while (i.hasNext()) {
   Service service = (Service) i.next();
   if (service.getQName() != null) {
    Map ports = service.getPorts();
    if (ports != null) {
     Iterator it = ports.values().iterator();
     while (it.hasNext()) {
      wsdlHash=new Hashtable();
      Port port = (Port) it.next();

      SOAPAddress sa = (SOAPAddress)
port.getExtensibilityElements().get(0);
      String SOAPURL = sa.getLocationURI();
      wsdlHash.put("LocationURI", sa.getLocationURI());

      Binding bind = (Binding) port.getBinding();
      BindingOperation soapop = (BindingOperation)
bind.getBindingOperations().get(0);
      String URI = ((SOAPBody)
soapop.getBindingInput().getExtensibilityElements().get(0)).getNa
mespaceURI();
      wsdlHash.put("NamespaceURI", URI);

      List opList =
port.getBinding().getPortType().getOperations();
```

```
       for (int j = 0; j < opList.size(); j++) {
        params = new Vector();
        Operation operation = (Operation) opList.get(j);
        String opr = operation.getName();
        System.out.println("\nOperation:" + opr);

        List inputPartsMap =
operation.getInput().getMessage().getOrderedParts(null);

        QName inPartTypeName = null;
        for(int x=0; x<inputPartsMap.size(); x++) {
         Part part = (Part) inputPartsMap.get(x);
         String inPartName = part.getName();
                                inPartTypeName =
part.getTypeName();
                                System.out.println(inPartName +
":" + inPartTypeName.getLocalPart());
         params.add(inPartTypeName.getLocalPart() + " " +
inPartName);
        }

        Map outputPartsMap =
operation.getOutput().getMessage().getParts();
        Collection outputParts = outputPartsMap.values();
        Iterator outputPartIter = outputParts.iterator();
        System.out.print("Response: ");
        QName outPartTypeName = null;
        while (outputPartIter.hasNext()) {
         Part part = (Part) outputPartIter.next();
         String outPartName = part.getName();
         outPartTypeName = part.getTypeName();
         System.out.println(outPartName + ":" +
outPartTypeName.getLocalPart());
         params.add(outPartTypeName.getLocalPart() + " " +
outPartName);
        }
        wsdlHash.put(opr, params);
       }
      }
     }
    }
   }
  return wsdlHash;
 }
}
```

```
package WSMS;
```

```
import net.n3.nanoxml.*;
import java.io.*;
import java.util.*;

/**
 *  Description of the Class
 *
 * @author     Mohini Padhye
 * @created    September 17, 2004
 */
public class StoreWS {

 /**
  *  This method stores the WSDL details in an XML file
  *
  * @param  wsdlUrl     url of the wsdl
  * @param  methodHash  Hashtable containing method and parameter
details
  */
 public void store(String wsdlUrl, Hashtable methodHash) {
  int wsid = 0;
  String baseDir = "Wrapper/"
  try {
   /* This method uses nanoXML API for XML parsing */
   IXMLParser parser = XMLParserFactory.createDefaultXMLParser();

   String storageFile = getWSName(wsdlUrl)+".xml";
   File sFile = new File(storageFile);

   if (!sFile.exists()) {
    createFile(baseDir+storageFile);
   }

   IXMLReader reader =
StdXMLReader.fileReader(baseDir+storageFile);
   parser.setReader(reader);

   IXMLElement xml = (IXMLElement) parser.parse();

   Writer output = (Writer) new FileWriter(baseDir+storageFile);

   for (int i = 0; i < xml.getChildrenCount(); i++) {
    String wsId = ((IXMLElement)
xml.getChildAtIndex(i)).getAttribute("wsid");
    wsid++;
   }
   wsid++;

   IXMLElement ws = new XMLElement("WS");
```

```
    ws.setAttribute("wsid", (new Integer(wsid)).toString());
    xml.addChild(ws);

    IXMLElement wsdlUrl1 = new XMLElement("WsdlUrl");
    wsdlUrl1.setContent(wsdlUrl);
    ws.addChild(wsdlUrl1);

    IXMLElement wsName = new XMLElement("WSName");
    wsName.setContent(getWSName(wsdlUrl));
    ws.addChild(wsName);

    Enumeration methodEnum = methodHash.keys();

    while (methodEnum.hasMoreElements()) {
     String methodElem = methodEnum.nextElement().toString();
     if (methodElem.equals("LocationURI")) {
      IXMLElement locURI = new XMLElement("LocationURI");

  locURI.setContent(methodHash.get("LocationURI").toString());
      ws.addChild(locURI);
     } else if (methodElem.equals("NamespaceURI")) {
      IXMLElement nsURI = new XMLElement("NamespaceURI");

  nsURI.setContent(methodHash.get("NamespaceURI").toString());
      ws.addChild(nsURI);
     }
    }

    methodEnum = methodHash.keys();
    int mid = 1;
    while (methodEnum.hasMoreElements()) {
     String methodElem = methodEnum.nextElement().toString();
     if(!methodElem.equals("LocationURI") &&
!methodElem.equals("NamespaceURI")) {
      String methodName = methodElem;

      IXMLElement method = new XMLElement("Method");
      method.setAttribute("mid", (new Integer(mid)).toString());
      mid++;
      ws.addChild(method);

      IXMLElement methodName1 = new XMLElement("MethodName");
      methodName1.setContent(methodName);
      method.addChild(methodName1);

      Vector paramVector = new Vector();
      paramVector = (Vector) methodHash.get(methodName);

      for (int i = 0; i < paramVector.size() - 1; i++) {
```

```
      IXMLElement paramType = new XMLElement("ParamType");
      paramType.setContent(paramVector.elementAt(i).toString());
      method.addChild(paramType);
     }

     IXMLElement returnType = new XMLElement("ReturnType");

 returnType.setContent(paramVector.elementAt(paramVector.size() -
1).toString());
     method.addChild(returnType);
    }
   }
   /* Writing to the XML document */
   XMLWriter writer = new XMLWriter(output);
   writer = new XMLWriter(output);
   writer.write(xml);

      System.out.println("\nService successfully added");

  } catch (Exception e) {
   e.printStackTrace();
  }
 }


 /**
  *  Gets the Web Service name from the wsdl url
  *
  * @param  wsdlUrl  Description of the Parameter
  * @return          The wSName value
  */
 public String getWSName(String wsdlUrl) {
  int startIndex = wsdlUrl.lastIndexOf("/");
  int endIndex = wsdlUrl.lastIndexOf(".");
  String wsName = wsdlUrl.substring(startIndex + 1, endIndex);
  return wsName;
 }


 /**
  *  Creates the XML data file
  *
  * @param  fileName        Description of the Parameter
  * @exception  IOException  Description of the Exception
  */
 public void createFile(String fileName) throws IOException {
  File storageFile = new File(fileName);
  storageFile.createNewFile();
  Writer output = (Writer) new FileWriter(fileName);
```

```
  XMLElement root = new XMLElement("WebService");
  XMLWriter writer = new XMLWriter(output);
  writer = new XMLWriter(output);
  writer.write(root);
 }
}
```

```
package WSMS;

import net.n3.nanoxml.*;
import java.io.*;
import java.util.*;

/**
 *  This class facilitates the user to view the details for
stores Web Service
 *
 * @author     Mohini Padhye
 * @created    September 17, 2004
 */
public class ViewWS {
 String baseDir = "Wrapper/"
 /**
  *  Details of the stored Web Service in terms of method names,
parameter names
  *  and data types can be viewed using this method
  *  This method uses nanoXML API for sml parsing
  * @exception  Exception  XML parsing exception
  */
 public void view() throws Exception {
  BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
  System.out.println("Enter Web Service Name:");
  String wsName = br.readLine();
  IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
  /* Opening the XML storage file for parsing */
  File storageFile = new File(baseDir+wsName+".xml");
   if (!storageFile.exists()) {
    System.out.println("XML Data file does not exist");
    return;
   }
  IXMLReader reader =
StdXMLReader.fileReader(baseDir+wsName+".xml");

  parser.setReader(reader);
  IXMLElement root = (IXMLElement) parser.parse();

  IXMLElement WSChild2 = (IXMLElement) root.getChildAtIndex(0);
```

```
  System.out.println(WSChild2.getAttribute("wsid") + ". WS Name
:" +
        (WSChild2.getChildAtIndex(1)).getContent());
  System.out.println("\nWSDL Url:" +
(WSChild2.getChildAtIndex(0)).getContent());

  /* Retrieving method and parameter names from XML storage */
  for (int j = 0; j < WSChild2.getChildrenCount(); j++) {
   IXMLElement childrenMethod = WSChild2.getChildAtIndex(j);
   for (int k = 0; k < childrenMethod.getChildrenCount(); k++) {
    if (k == 0) {
     System.out.println("\n" +
childrenMethod.getAttribute("mid") + ". Method Name:" +
      ((IXMLElement)
childrenMethod.getChildAtIndex(0)).getContent() + "\n");
    } else if (k < childrenMethod.getChildrenCount() - 1) {
     System.out.println("Input Param:" +
                              (IXMLElement)
      childrenMethod.getChildAtIndex(k)).getContent());
    } else {
     System.out.println("\nReturn Param:" +
                              ((IXMLElement)
      childrenMethod.getChildAtIndex(k)).getContent() + "\n");
    }
   }
  }
 }
}
```

```
package Wrapper;

import net.n3.nanoxml.*;
import java.io.*;

/**
 *  Description of the Class
 *
 * @author     Mohini Padhye
 * @created    September 17, 2004
 */
public class CreateBond {
 String srcWS = "";
 String baseDir = "Wrapper/"
 /**
  *  This method creates bonds between two web service methods
  *  It takes source and destonation web service name as well as
  *  method name as parametes from user along with type of bond
and
```

```
  *  presence of trigger for creating the bond. It stores the
information
  *  in the persistent XML storage
  */

 public void createWebBond() {
  try {
   BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));

   System.out.println("Enter Source Web Service Name :");
   srcWS = br.readLine();
   System.out.println("Enter Source Method Name :");
   String srcMethod = br.readLine();
   System.out.println("Enter Destination Web Service Name :");
   String destWS = br.readLine();
   System.out.println("Enter Destination method Name:");
   String destMethod = br.readLine();
   System.out.println("Enter Bond Type(S/N):");
   String bondType = br.readLine();
   System.out.println("Do you want to create a trigger for this
bond(Y/N):");
   String trigger = br.readLine();
   System.out.println("If this is a boolean bond, enter the
operator(AND/OR/XOR), else NO:");
      String booleanOpr = br.readLine();

   IXMLParser parser = XMLParserFactory.createDefaultXMLParser();

   String fileName="";
   if(bondType.equals("S"))
    fileName = srcWS + ".xml";
   else if(bondType.equals("N")) {
    fileName = destWS + ".xml";
    String temp="";
    temp=srcWS;
    srcWS=destWS;
    destWS=temp;

    temp=srcMethod;
    srcMethod=destMethod;
    destMethod=temp;
   }

   File wrapperFile = new File(baseDir+fileName);

   IXMLReader reader = StdXMLReader.fileReader(baseDir+fileName);
   parser.setReader(reader);
```

```
    IXMLElement root = (IXMLElement) parser.parse();
    int childCount = root.getChildrenCount();

    IXMLElement wrapper = null;
    if(childCount <= 1) {
     wrapper = (IXMLElement)new XMLElement("Wrapper");
     root.addChild(wrapper);
    } else {
     wrapper = root.getChildAtIndex(1);
    }

    int wrapperChildCount = wrapper.getChildrenCount();

    int bid = 1;
    if (wrapperChildCount > 1) {
     for (int i = 1; i < wrapper.getChildrenCount(); i++) {
      String bId = ((IXMLElement)
wrapper.getChildAtIndex(i)).getAttribute("bid");
      bid++;
     }
     bid++;
    }

    /* Stroring the bond information in XML file */
    IXMLElement bond = new XMLElement("Bond");
    bond.setAttribute("bid", (new Integer(bid)).toString());
    wrapper.addChild(bond);

    IXMLElement srcM = new XMLElement("SrcMethod");
    srcM.setContent(srcMethod);
    bond.addChild(srcM);

    IXMLElement destW = new XMLElement("DestWS");
    destW.setContent(destWS);
    bond.addChild(destW);

    IXMLElement destM = new XMLElement("DestMethod");
    destM.setContent(destMethod);
    bond.addChild(destM);

    IXMLElement type = new XMLElement("Type");
    type.setContent(bondType);
    bond.addChild(type);

    IXMLElement trig = new XMLElement("Trigger");
    trig.setContent(trigger);
    bond.addChild(trig);

    Writer output = (Writer) new FileWriter(baseDir+fileName);
```

```
    XMLWriter writer = new XMLWriter(output);
    writer = new XMLWriter(output);
    writer.write(root);

    System.out.println("\nBond created successfully");

   } catch (FileNotFoundException e) {
    System.out.println("\nBond creation failed");
    System.out.println(e.toString());
   } catch (Exception e) {
    e.printStackTrace();
   }
  }
}
```

```
package Wrapper;

import net.n3.nanoxml.*;
import java.io.*;

/**
 *  Bond deletion is achieved through this method
 * @author     Mohini Padhye
 * @created    September 21, 2004
 */
public class DeleteBond {
 String baseDir = "Wrapper/";
 /**
  *  This method details of a bond from XML storage
  */
 public void deleteWebBond() {
  try {
   System.out.println("Please enter the Web Service name:");
   BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
   String nameWS = br.readLine();

   System.out.println("Please enter (serial number)/id of the
bond to be deleted:");
   String bid = br.readLine();

   IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
   String fileName = baseDir+nameWS + ".xml";
   IXMLReader reader = StdXMLReader.fileReader(fileName);
   parser.setReader(reader);

   IXMLElement docroot = (IXMLElement) parser.parse();
   IXMLElement root = (IXMLElement) docroot.getChildAtIndex(1);
```

```
    for (int i = 1; i < root.getChildrenCount(); i++) {
     IXMLElement bond = root.getChildAtIndex(i);
     /* Bond deletion, the node containing bond information is
deleted */
     if (bond.getAttribute("bid").equals(bid)) {
      root.removeChild(bond);
      System.out.println("Bond Deleted Successfully");
     }
    }

    Writer output = (Writer) new FileWriter(fileName);
    XMLWriter writer = new XMLWriter(output);
    writer = new XMLWriter(output);
    writer.write(docroot);

   } catch (Exception e) {
    System.out.println(e.toString());
   }
  }
}
```

```
package Wrapper;

import net.n3.nanoxml.*;
import java.io.*;

/**
 *  This method facilitates the user to view the bond information
 *
 * @author     Mohini Padhye
 * @created    September 22, 2004
 */
public class ViewBond {
 String baseDir = "Wrapper/"
 /**
  *  Bond information is shown by this method in the form of
  *  source and destination web service as well as method names,
  * type of bond etc
  */
 public void viewWebBond() {
  try {
   /* Taking user input */
   System.out.println("Please enter the Web Service name:");
   BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
   String nameWS = br.readLine();
```

```
    IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
    IXMLReader reader = StdXMLReader.fileReader(baseDir+nameWS +
".xml");
    parser.setReader(reader);
    IXMLElement docroot = (IXMLElement) parser.parse();
    IXMLElement root = (IXMLElement) docroot.getChildAtIndex(1);

    System.out.println("Source Web Service :" + nameWS);
    for (int i = 0; i < root.getChildrenCount(); i++) {
     IXMLElement bond = (IXMLElement) root.getChildAtIndex(i);
     System.out.println(bond.getAttribute("bid") + ". Web Bond");
     System.out.println("Source Method:" +
bond.getChildAtIndex(0).getContent());
     System.out.println("Destination Web Service:" +
bond.getChildAtIndex(1).getContent());
     System.out.println("Destination Method:" +
bond.getChildAtIndex(2).getContent());
     String type = bond.getChildAtIndex(3).getContent();

     if (type.equals("S")) {
      System.out.println("Bond Type: Subscription Bond");
     } else {
      System.out.println("Bond Type: Negotiation Bond");
     }

     String trigger = bond.getChildAtIndex(4).getContent();

     if (trigger.equals("Y")) {
      System.out.println("Trigger : Yes");
     } else {
      System.out.println("Trigger : No");
     }
    }

   } catch (Exception e) {
    System.out.println("Exception :" + e.toString());
   }
  }
}
```

```
package Wrapper;

import java.io.*;
import java.util.*;
import java.net.*;
import java.lang.*;
import java.lang.reflect.*;
import net.n3.nanoxml.*;
```

```
/**
 *  This class deals with all the operations
 *  on the wrapper inclusing creating, executing, checking
 *   a wrapper
 *  For every web service, two files are created, viz,
__wsName__.java and
 *  __wsName__Impl.java. Both these files are SyD enabled.
__wsName__.java file
 *  is an interface while __wsName__Impl.java is its
implementation
 * Both the files are RMI enabled, which is a specification for
SyD for
 * SySD directory and remote object invocation
 * @author     Mohini Padhye
 * @created    September 19, 2004
 */

 public class Wrapper {
 DataOutputStream out,outImpl;
 String wrapperFile = "",implFile="";
 Hashtable initData = new Hashtable();
 String baseDir="Wrapper/";

 /**
  *  This method creates the wrapper for a web service
  *
  * @param  wsdlUrl     url for the wsdl file
  * @param  methodHash  hashtable containing method and parameter
list
  */
 public void createWrapper(String wsdlUrl, Hashtable methodHash)
{
  try {
   String fileName = getWrapperName(wsdlUrl);
   implFile =
fileName.substring(0,fileName.lastIndexOf("."))+"Impl";
   out = new DataOutputStream(new
FileOutputStream(baseDir+implFile+".java"));
   outImpl = new DataOutputStream(new
FileOutputStream(baseDir+fileName));
   initValues();
   createDefaultCode();
   createWSCode(wsdlUrl, methodHash);
   out.close();
   outImpl.close();
   Runtime.getRuntime().exec("javac "+baseDir+fileName);
   Runtime.getRuntime().exec("javac "+baseDir+implFile);
  } catch (Exception e) {
```

```
    System.out.println(e.toString());
   }
  }


  /**
   *  This method creates the default code for the wrapper files
   */
  private void createDefaultCode() {
   try {
    out.writeBytes("package syd.sydapp;\n\n"+
           "import java.io.*;\nimport java.net.*;\nimport
java.util.*;\n"+
          "import java.lang.reflect.*;\nimport
net.n3.nanoxml.*;\n\n"+
          "import org.apache.axis.client.*;\nimport
org.apache.axis.encoding.*;"+
           "\n\nimport java.rmi.RemoteException;\n"+
          "import java.rmi.server.UnicastRemoteObject;\n"+
           "\nimport org.apache.axis.utils.*;\n"+
          "import javax.xml.namespace.QName;\n"+
           "import javax.xml.rpc.ParameterMode;\n\n");
    out.writeBytes("public class " + implFile + " extends
UnicastRemoteObject implements "+wrapperFile+" {\n\n");
    out.writeBytes("\tString baseDir=\"syd/sydapp\";\n\tpublic " +
implFile + "() throws RemoteException {\n\t\tsuper();\n\t}\n\n");

    outImpl.writeBytes("package syd.sydapp;\n\nimport
java.rmi.Remote;\nimport java.rmi.RemoteException;\n");
            outImpl.writeBytes("\npublic interface " +
wrapperFile+ " extends Remote{\n");
   } catch (Exception e) {
    System.out.println(e.toString());
   }

  }

  /**
   *  This method creates web service specific code
   *
   * @param  wsdlUrl     url of the wsdl file
   * @param  methodHash  hashtable containg method and parameter
list
   */
  private void createWSCode(String wsdlUrl, Hashtable methodHash)
{
   try {
    String locURI = "";
    String nsURI = "";
```

```
   int loopCnt = 0;

   Enumeration methodEnum = methodHash.keys();

   while (methodEnum.hasMoreElements()) {
    String methodElem = methodEnum.nextElement().toString();
    if (methodElem.equals("LocationURI")) {
     locURI = methodHash.get(methodElem).toString();
    } else if (methodElem.equals("NamespaceURI")) {
     nsURI = methodHash.get(methodElem).toString();
    }
   }

   methodEnum = methodHash.keys();
   while (methodEnum.hasMoreElements()) {
    String methodElem = methodEnum.nextElement().toString();
    if( !methodElem.equals("LocationURI") &&
!methodElem.equals("NamespaceURI")) {
     String methodName = methodElem;
     Vector paramVector = new Vector();
     paramVector = (Vector) methodHash.get(methodName);

     String rVar = paramVector.elementAt(paramVector.size() -
1).toString();
     String returnVar = rVar.substring(0, rVar.lastIndexOf("
"));

     String inputParam = "";
     String inputValues = "";

     String ret = paramVector.elementAt(paramVector.size() -
1).toString();
     ret = ret.substring(0, ret.lastIndexOf(" "));

     if (ret.equals("string")) {
      ret = "String";
     } else if (ret.equals("date")) {
      ret = "Date";
     }

     out.writeBytes("\tpublic " + ret + " " + methodName + "(");
     outImpl.writeBytes("\tpublic " + ret + " " + methodName +
"(");

     for (int i = 0; i < paramVector.size() - 1; i++) {
      String input = paramVector.elementAt(i).toString();

      String type = input.substring(0, input.lastIndexOf(" "));
```

```
       Properties ObjectData = new Properties();
            ObjectData.load(new
FileInputStream(baseDir+ObjectData.properties"));
       type = ObjectData.getProperty(type);

       String value = input.substring(input.lastIndexOf(" ") + 1,
input.length());
       inputParam += type + " " + value + ",";
      }

      inputParam = inputParam.substring(0, inputParam.length() -
1);

      outImpl.writeBytes(inputParam + ") throws
RemoteException;\n");

      out.writeBytes(inputParam + ") throws RemoteException {"+
            "\n\t\tString methodName=\"" + methodName + "\";"+
            "\n\t\tString wrapperFile=baseDir+\"/" + wrapperFile
+ ".xml\";"+
            "\n\t\t" + ret + "
retVal="+initData.get(ret).toString()+";"+
            "\n\n\t\ttry {"+
                            "\n\t\t\tFile wrapperF = new
File(wrapperFile);"+
                            "\n\t\t\tif(wrapperF.exists()) {"+
                            "\n\t\t\t\tboolean
isNegBond=this.checkForBonds(methodName,wrapperFile,\"N\");");
                      out.writeBytes("\n\t\t\t}"+
          "\n\t\t\tService service = new Service();"+
          "\n\t\t\tCall call = (Call) service.createCall();"+
          "\n\t\t\tcall.setTargetEndpointAddress(new
java.net.URL(\"" + locURI + "\"));"+
          "\n\t\t\tcall.setOperationName(new QName(\"" + nsURI +
"\",\"" + methodName + "\"));");

      Properties XSDData = new Properties();
      XSDData.load(new
FileInputStream(baseDir+"XSDData.properties"));

          Properties ObjectData = new Properties();
          ObjectData.load(new
FileInputStream(baseDir+"ObjectData.properties"));

      for (int i = 0; i < paramVector.size() - 1; i++) {
       String param = paramVector.elementAt(i).toString();
```

```
       String param1 =
param.substring((param.trim().lastIndexOf(" ") + 1),
(param.length()));

       String dataType = param.substring(0, param.lastIndexOf("
"));

       String param2 = XSDData.getProperty(dataType);

       inputValues += param1 + ",";

       out.writeBytes("\n\t\t\tcall.addParameter(\"" + param1 +
"\",XMLType." + param2 + ",ParameterMode.IN);");
      }
      inputValues = inputValues.substring(0, inputValues.length()
- 1);
      out.writeBytes("\n\t\t\tcall.setReturnType(XMLType." +
XSDData.get(returnVar).toString() + ");");

      if (returnVar.equals("int")) {
       out.writeBytes("\n\t\t\tretVal = ((Integer)
call.invoke(new Object[]{" + inputValues + "})).intValue();");
      }
      if (returnVar.equals("float")) {
       out.writeBytes("\n\t\t\tretVal = ((Float) call.invoke(new
Object[]{" + inputValues + "})).floatValue();");
      }
      if (returnVar.equals("double")) {
       out.writeBytes("\n\t\t\tretVal = ((Double) call.invoke(new
Object[]{" + inputValues + "})).doubleValue();");
      }
      if (returnVar.equals("boolean")) {
       out.writeBytes("\n\t\t\tretVal = ((Boolean)
call.invoke(new Object[]{" + inputValues +
"})).booleanValue();");
      }
      if (returnVar.equals("string")) {
       out.writeBytes("\n\t\t\tretVal = (String) call.invoke(new
Object[]{" + inputValues + "});");
      }

      out.writeBytes("\n\t\t\tSystem.out.println(\"Return
Value:\"+retVal);"+
          "\n\t\t\tif(wrapperF.exists()) {"+
          "\n\t\t\t\tboolean
isSubBond=this.checkForBonds(methodName,wrapperFile,\"S\");"+
          "\n\t\t\t\tif(isSubBond) {"+
```

```
 "\n\t\t\t\t\tthis.executeSubscriptionBond(methodName,wrapperFile
);"+
            "\n\t\t\t\t}"+
            "\n\t\t\t}"+
          "\n\t\t} catch(Exception e)
{System.out.println(e.toString());}"+
          "\n\t\treturn retVal;"+
          "\n\t}\n\n");
    }
    loopCnt++;
   }
   this.writeCheckForBonds();
   this.writeExecuteSubscriptionBond();
   out.writeBytes("\n}");
   outImpl.writeBytes("}");
  } catch (Exception e) {
   System.out.println(e.toString());
  }
 }

 /**
  *  Retrieved name of the wrapper file from wsdl url
  *  name of the wrapper is same as name of the web service
  * @param  wsdlUrl  url of the wsdl file
  * @return          name of the wrapper
  */
 private String getWrapperName(String wsdlUrl) {
  int startIndex = wsdlUrl.lastIndexOf("/");
  int endIndex = wsdlUrl.lastIndexOf(".");

  wrapperFile = wsdlUrl.substring(startIndex + 1, endIndex);
  return wrapperFile + ".java";
 }

 /**
  * This method writes the executeSubscriptionBond method to the
wrapper file
  * @param  methodName     Source method name
  * @param  wFile          Name of the wrapper file
  */
 public void writeExecuteSubscriptionBond() {
  try {
   out.writeBytes("\n\tpublic void executeSubscriptionBond(String
methodName, String wFile) throws Exception {"+
   "\n\t\ttry { "+
   "\n\t\t boolean hasBond=false; "+
   "\n\t\t String destWS=\"\";"+
   "\n\t\t String destMethod=\"\";"+
```

```
    "\n\t\t String srcMethod;"+
    "\n\t\tVector paramVector = new Vector(); "+
    "\n\t\tVector paramType = new Vector(); "+
    "\n\t\tIXMLParser parser =
XMLParserFactory.createDefaultXMLParser(); "+
    "\n\t\tIXMLReader reader = StdXMLReader.fileReader(wFile); "+
    "\n\t\tparser.setReader(reader); "+
    "\n\t\tIXMLElement fileRoot = (IXMLElement) parser.parse(); "+
    "\n\t\tIXMLElement root = (IXMLElement)
fileRoot.getChildAtIndex(1); "+
    "\n\t\tfor (int i = 0; i < root.getChildrenCount(); i++) { "+
    "\n\t\t\t IXMLElement bond = (IXMLElement)
root.getChildAtIndex(i); "+
    "\n\t\t\t for (int j = 0; j < bond.getChildrenCount(); j++) {
"+
    "\n\t\t\t\t if ((((IXMLElement)
bond.getChildAtIndex(j++)).getContent()).equals(methodName)) { "+
    "\n\t\t\t\t\t destWS = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent(); "+
    "\n\t\t\t\t\t destMethod = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent(); "+
    "\n\t\t\t\t\t String bondType = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent(); "+
    "\n\t\t\t\t\t String trigger = ((IXMLElement)
bond.getChildAtIndex(j++)).getContent(); "+
    "\n\t\t\t\t\t if (bondType.equals(\"S\") &&
trigger.equals(\"Y\")) { "+
    "\n\t\t\t\t\t\t   hasBond=true; "+
    "\n\t\t\t\t\t\t   WrapperAppO wAppO = new WrapperAppO();"+
    "\n\t\t\t\t\t\t   paramType =
wAppO.showDestParamInputFrame(destWS,destMethod);"+
    "\n\t\t\t\t\t\t   paramVector = wAppO.getDestParams();"+
    "\n\t\t\t\t\t   } "+
    "\n\t\t\t\t  } "+
    "\n\t\t\t } "+
    "\n\t\t\t  if (hasBond == false) { "+
    "\n\t\t\t\t   return; "+
    "\n\t\t\t  } else { "+
    "\n\t\t\t\t   WrapperAppO wAppO = new WrapperAppO(); "+
    "\n\t\t\t\t   String retVal=wAppO.invoke(destWS, destMethod,
paramType, paramVector); "+
    "\n\t\t\t\t   System.out.println(\"Return:\"+retVal);"+
    "\n\t\t\t  } "+
    "\n\t\t\t  hasBond = false; "+
    "\n\t\t } "+
    "\n\t\t} catch (Exception e) { "+
    "\n\t\t\t e.printStackTrace(); "+
    "\n\t\t} "+
  "\n\t} ");
```

```
  } catch (Exception e) {
   e.printStackTrace();
  }
 }

 /**
  * This method writes the writeCheckForBonds method to the
wrapper file
  */
 public void writeCheckForBonds() {
  try {
   out.writeBytes("\n\n \tpublic boolean checkForBonds(String
methodName,String wrapperFile,String bondType) { "+
   " \n\t  String bType; "+
   "\n\t try { "+
   "\n\t\t File wrapperF = new File(wrapperFile); "+
   "\n\t\t    if(!wrapperF.exists()) {"+
   "\n\t\t\t    System.out.println(\"XML data file does not
exist\");"+
   "\n\t\t\t        return false; "+
   "\n\t\t    }"+
   "\n\t\t    IXMLParser parser =
XMLParserFactory.createDefaultXMLParser(); "+
   "\n\t\t    IXMLReader reader =
StdXMLReader.fileReader(wrapperFile); "+
   "\n\t\t    parser.setReader(reader); "+
   "\n\t\t    IXMLElement mainRoot = (IXMLElement)
parser.parse(); "+
   "\n\t\t    if(mainRoot.getChildrenCount() > 1) {"+
   "\n\t\t\t    IXMLElement root = (IXMLElement)
mainRoot.getChildAtIndex(1); "+
   "\n\t\t\t    for (int i = 0; i < root.getChildrenCount(); i++)
{ "+
   "\n\t\t\t\t      IXMLElement bond = (IXMLElement)
root.getChildAtIndex(i); "+
   "\n\t\t\t\t      for (int j = 0; j < bond.getChildrenCount();
j++) { "+
   "\n\t\t\t\t\t        bType = ((IXMLElement)
bond.getChildAtIndex(3)).getContent(); "+
   "\n\t\t\t\t\t    if(bType.equals(bondType)) { "+
   "\n\t\t\t\t\t\t    return true; "+
   "\n\t\t\t\t\t    } "+
   "\n\t\t\t\t  } "+
   "\n\t\t\t } "+
   "\n\t\t } "+
   "\n\t } catch (Exception e) { "+
   "\n\t\t  e.printStackTrace(); "+
   "\n\t } "+
   "\n\t return false; "+
```

```
    "\n\t} \n\n");
   } catch (Exception e) {
      e.printStackTrace();
     }
  }
}
```

```
package syd.sydapp;

import java.io.*;

import java.awt.*;
import java.awt.event.*;

import java.rmi.*;
import java.rmi.registry.*;

import java.util.*;
import java.lang.*;
import java.lang.reflect.*;
import syd.sydlistener.*;
import syd.sydapp.*;
import syd.sydutil.*;
import syd.sydengine.*;
import syd.syddirectory.*;
import java.net.*;

import net.n3.nanoxml.*;

/**
 * This class works with SyD Wrapper objects (generated per web
service)
 * to integrate the wrapper with the SyD environment
 * @author     Mohini Padhye
 * @created    November 11, 2004
 */
public class WrapperAppO extends Frame implements ActionListener
{
 private int PORT = 8000;
 private static String myurl = null;
 Vector methodParams = new Vector();
 Vector params = new Vector();
 Vector methods = new Vector();
 Vector returnTypes = new Vector();
 String sydDir = "syd/sydapp/";
 String baseDir = "syd.sydapp.";
```

```
 /**
  *  This method does the task of invoking a remote object using
  *  SyDDispatcher class of SyDEngine
  *
  * @param  objId       object id of the wrapper that has been
registered with
  *           the SyDDirectory
  * @param  methodName  Name of the method to be invoked
  * @param  paramType   Data type of the parammeter in a vector
  * @param  paramValue  Actual data values in a vector
  * @return             result string
  */
 public String invoke(String objId, String methodName, Vector
paramType, Vector paramValue) {
  String outputValue = "";
  try {
   SyDDispatcher dispatcher = new SyDDispatcher();
   outputValue = dispatcher.invoke(objId, methodName, paramType,
paramValue);
   dispose();
   this.displayMessage("Result of " + objId + "," + methodName +
":" + outputValue);
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return outputValue;
 }


 /**
  *  Method to register a SyD Wrapper with the SyDDirectory
  *
  * @param  wsName               Name of the web service
  * @param  portNum              local port number
  * @param  flag                 flag
  * @param  directoryServerName  directory server name
  * @param  directoryServerPort  directory server port
  */
 public void registrate(String wsName, int portNum, int flag,
String directoryServerName, int directoryServerPort) {
  getParamMethods(wsName);
  try {
   myurl = InetAddress.getLocalHost().getHostAddress();
   Publisher publisher = new Publisher();
   String userID = wsName;
   String userPasswd = wsName;
   String userURL = myurl;
   String proxyID = "121";
   String appName = wsName;
```

```
   /* Web service information is initially converted into
required XML format*/
   publisher.createPublishUserMethodsRequest(userID, userPasswd,
userURL, proxyID, appName, methods, returnTypes, params);

   String xmlDoc = publisher.getString();
   System.out.println("XML Doc:" + xmlDoc);

   try {
    if (flag == 1) {
     System.out.println("Creating RMI registry at port " +
portNum + "...");
     LocateRegistry.createRegistry(portNum);
     System.out.println("RMI Registry created at " + portNum +
".");
    }
    System.out.println("Constructing server implementations...");

    Class wrapperClass = Class.forName(baseDir + wsName +
"Impl");
    Object wrapperObj = wrapperClass.newInstance();

    System.out.println("Binding server implementations to
registry...");
    SyDRegistrar registrar = new SyDRegistrar("localhost",
portNum, directoryServerName, directoryServerPort);
    /* Registering the service */
    registrar.register(wrapperObj, xmlDoc);
    System.out.println("objectName: " +
registrar.getObjectName());
    System.out.println("Waiting for invocations from
clients...");
   } catch (Exception e) {
    System.out.println(e.toString());
   }
  } catch (java.net.UnknownHostException e) {
   System.out.println(e.toString());
  }
 }


 /**
  *  This method acts as an interface between the front end and
registrate method
  *
  * @param  wsName  Name of web service
  */
 public boolean createWrapperBinding(String wsName) {
  try {
```

```
    SyDPropertyFile p = new SyDPropertyFile();
    String dirurl = p.getValue("sydprop", "directoryurl");
    String listenerurl = p.getValue("sydprop", "listenerurl");
    /* regustrate method invoked */
    registrate(wsName, PORT, 1, dirurl, 1099);
    return true;
   } catch (Exception e) {
    System.out.println(e.getMessage());
    return false;
   }
  }


  /**
   *  Gets the list of methods and parameters for a web service
   *
   * @param  wsName  Name of the web service
   */
  public void getParamMethods(String wsName) {
   try {
    /* retrieves data from XML storage file */
    IXMLParser parser = XMLParserFactory.createDefaultXMLParser();
    File storageFile = new File(sydDir + wsName + ".xml");

    if (!storageFile.exists()) {
     System.out.println("XML Data file does not exist");
     return;
    }

    IXMLReader reader = StdXMLReader.fileReader(sydDir + wsName +
".xml");
    parser.setReader(reader);
    IXMLElement root = (IXMLElement) parser.parse();
    IXMLElement WSChild2 = (IXMLElement) root.getChildAtIndex(0);

    Properties objectData = new Properties();
    objectData.load(new FileInputStream(sydDir +
"JavaDataTypes.properties"));

    for (int j = 0; j < WSChild2.getChildrenCount(); j++) {
     IXMLElement childrenMethod = WSChild2.getChildAtIndex(j);
     if (childrenMethod.getName().equals("Method")) {
      for (int k = 0; k < childrenMethod.getChildrenCount(); k++)
{
       if (k == 0) {
        methods.addElement(((IXMLElement)
childrenMethod.getChildAtIndex(0)).getContent());
       } else if (k < childrenMethod.getChildrenCount() - 1) {
```

```
        String inputParam = ((IXMLElement)
childrenMethod.getChildAtIndex(k)).getContent();
        inputParam = inputParam.substring(0,
inputParam.lastIndexOf(" "));
        inputParam = objectData.getProperty(inputParam);
        methodParams.addElement(inputParam);
       } else {
        String returnVal = ((IXMLElement)
childrenMethod.getChildAtIndex(k)).getContent();
        returnVal = returnVal.substring(0,
returnVal.lastIndexOf(" "));
        returnVal = objectData.getProperty(returnVal);
        returnTypes.addElement(returnVal);
       }
      }
      params.addElement(methodParams);
      methodParams = new Vector();
     }
    }
  } catch (Exception e) {
   System.out.println(e.toString());
  }
 }


 /**
  *  Gets list of parameter data types for a specified method of
the web service
  *
  * @param  WSName      web service name
  * @param  methodName  method name
  * @return             vector containing parameter data types
  */
 private Vector getParams(String WSName, String methodName) {
  Vector inputVector = new Vector();
  try {
   IXMLParser parser = XMLParserFactory.createDefaultXMLParser();

   File storageFile = new File(sydDir + WSName + ".xml");

   if (!storageFile.exists()) {
    System.out.println("XML Data file does not exist");
    return null;
   }

   IXMLReader reader = StdXMLReader.fileReader(sydDir + WSName +
".xml");

   parser.setReader(reader);
```

```
   IXMLElement root = (IXMLElement) parser.parse();

   IXMLElement WSChild2 = (IXMLElement) root.getChildAtIndex(0);
   Properties objectData = new Properties();
   objectData.load(new FileInputStream(sydDir +
"JavaDataTypes.properties"));

   for (int j = 0; j < WSChild2.getChildrenCount(); j++) {
    IXMLElement childrenMethod = WSChild2.getChildAtIndex(j);
    if (childrenMethod.getName().equals("Method")) {
     for (int k = 1; k < childrenMethod.getChildrenCount() - 1;
k++) {
      if ((((IXMLElement)
childrenMethod.getChildAtIndex(0)).getContent()).equals(methodNam
e)) {
       String inputParam = ((IXMLElement)
childrenMethod.getChildAtIndex(k)).getContent();
       inputParam = inputParam.substring(0,
inputParam.lastIndexOf(" "));
       inputParam =
objectData.getProperty(inputParam).toString();
       inputVector.addElement(inputParam);
      }
     }
    }
   }
  } catch (Exception e) {
   System.out.println(e.toString());
  }
  return inputVector;
 }


 /**
  *  Gets destination parameter data types from SyDDirectory
  *
  * @param  username    name of the web service, which is same as
username
  * @param  destMethod  name of destination web service
  * @return             destination parameter data types
  */
 public Vector getDestWSParamsFromDir(String username, String
destMethod) {
  Vector paramV = new Vector();
  try {
   SyDPropertyFile p = new SyDPropertyFile();
   String dirurl = p.getValue("sydprop", "directoryurl");
   String listenerurl = p.getValue("sydprop", "listenerurl");
```

```
    MemberShipI member1 = null;
    Registry r = LocateRegistry.getRegistry(dirurl);
    member1 = (MemberShipI) r.lookup("DirectoryService");

    String userid = member1.lookUp("SYD_USER", "userID",
"userName", username);
    String objectid = member1.lookUp("USER_APPO_MAPPING",
"objectID", "userID", userid);
    String url = member1.lookUp("SYD_USER", "userURL", "userName",
username);
    String paramList = member1.getParams(userid, objectid,
destMethod);

    StringTokenizer st = new StringTokenizer(paramList);
    while (st.hasMoreTokens()) {
     String tok = st.nextToken().toString();
     if ((tok.trim()).equals("String")) {
      tok = "java.lang.String";
     }
     paramV.add(tok);
    }
   } catch (Exception e) {
    System.out.println(e.toString());
   }
   return paramV;
  }


  /**
   *  Invokes the web service dynamically using java's reflection
API
   *
   * @param  wName      web service name
   * @param  mName      method name
   * @param  prmNames   parameter types
   * @param  prmValues  parameter values
   */
  public void callWSMethod(String wName, String mName, Vector
prmNames, Vector prmValues) {
   try {
    Vector paramDataType = new Vector();
    Vector inputData = new Vector();

    Properties objectData = new Properties();
    objectData.load(new FileInputStream(sydDir +
"JavaDataTypes.properties"));

    for (int i = 0; i < prmNames.size(); i++) {
```

```
     String paramTypes = prmNames.elementAt(i).toString();
     paramDataType.add(paramTypes);
    }

    for (int i = 0; i < prmValues.size(); i++) {
     String paramType = paramDataType.elementAt(i).toString();
     if (paramType.equals("java.lang.Integer")) {
      inputData.add(new
Integer(Integer.parseInt(prmValues.elementAt(i).toString())));
     }
     if (paramType.equals("java.lang.Float")) {
      inputData.add(new
Float(Float.parseFloat(prmValues.elementAt(i).toString())));
     }
     if (paramType.equals("java.lang.Double")) {
      inputData.add(new
Double(Double.parseDouble(prmValues.elementAt(i).toString())));
     }
     if (paramType.equals("java.lang.Boolean")) {
      inputData.add(new
Boolean(Boolean.getBoolean(prmValues.elementAt(i).toString())));
     }
     if (paramType.equals("java.lang.String")) {
      inputData.add(prmValues.elementAt(i));
     }
    }

    //Java Reflection, call to local web service
    Class wsClass = Class.forName(baseDir + wName + "Impl");
    Method[] wsMethods = wsClass.getMethods();
    Object retVal = null;
    for (int z = 0; z < wsMethods.length; z++) {
     if (wsMethods[z].getName().equals(mName)) {
      Vector objectParam = new Vector();
      for (int m = 0; m < inputData.size(); m++) {
       objectParam.add(inputData.elementAt(m));
      }
      Object object[] = objectParam.toArray();
      retVal = wsMethods[z].invoke(wsClass.newInstance(),
object);
      break;
     }
    }
    dispose();
   } catch (Exception e) {
    System.out.println(e.toString());
    e.printStackTrace();
   }
   return;
```

```
 }

/* UI part starts here */
 static Label ws_label = new Label("Enter WS Name:");
 static Label method_label = new Label("Enter method Name:");

 static TextField ws_txt = new TextField();
 static TextField method_txt = new TextField();

 static Checkbox reg_chkbx = new Checkbox("Register");
 static Checkbox invoke_chkbx = new Checkbox("Invoke");

 final static String cmd_ok = "Ok";
 final static String cmd_invoke = "Invoke";
 final static String cmd_register = "Register";
 final static String cmd_cancel = "Cancel";
 final static String cmd_next = "Next";
 final static String cmd_menu = "Menu";
 final static String cmd_proceed = "Proceed";

 static Button m_btn_ok = new Button(cmd_ok);
 static Button m_btn_menu = new Button(cmd_menu);
 static Button m_btn_cancel = new Button(cmd_cancel);
 static Button m_btn_proceed = new Button(cmd_proceed);
 static Button m_btn_next = new Button(cmd_next);
 static Button m_btn_invoke = new Button(cmd_invoke);


 /**
  *  Entry point of the class
  */
 public static void main(String[] args) {
  WrapperAppO frame = new WrapperAppO();
  frame.setVisible(true);
 }


 /**
  *Constructor for the WrapperAppO object
  */
 public WrapperAppO() {
  init();
 }


 /**
  *Constructor for the WrapperAppO object
  */
 public WrapperAppO(String status) { }
```

```
 /**
  *  Action performed method, to initiate action in component
changes
  * @param  e  Action event
  */
 public void actionPerformed(ActionEvent e) {
  if (cmd_ok.equals(e.getActionCommand())) {
   Vector paramValues = new Vector();
   for (int i = 0; i < paramTextField.length; i++) {
    paramValues.add(paramTextField[i].getText());
   }
   this.dispose();
   callWSMethod(getWSName(), getMethodName(), paramNames,
paramValues);
   this.removeAll();
   this.initComponents();
   init();
  }
  if (cmd_invoke.equals(e.getActionCommand())) {
   destValueVector = new Vector();
   for (int i = 0; i < destParamTextField.length; i++) {
    destValueVector.add(destParamTextField[i].getText());
   }
   inputDialog.dispose();
   this.initComponents();
   this.dispose();
  }
  if (cmd_cancel.equals(e.getActionCommand())) {
   dispose();
   System.exit(0);
  }
  if (cmd_menu.equals(e.getActionCommand())) {
   this.removeAll();
   this.initComponents();
   init();
  }
  if (cmd_proceed.equals(e.getActionCommand())) {
   setRegChkBxStatus(reg_chkbx.getState());
   setInvokeChkBxStatus(invoke_chkbx.getState());
   this.removeAll();
   this.initComponents();
   initInvokeMenu();
  }
  if (cmd_next.equals(e.getActionCommand())) {
   setWSName(ws_txt.getText());
   setMethodName(method_txt.getText());
   if (regChkBxStatus == true) {
```

```
      SyDPropertyFile p = new SyDPropertyFile();
      String dirurl = p.getValue("sydprop", "directoryurl");
      createWrapperBinding(getWSName());
      this.removeAll();
      displayMessage("Registration Successful");
     } else if (invokeChkBxStatus == true) {
      this.removeAll();
      showParamInputFrame();
     }
     this.initComponents();
    }
   }

   public Vector getDestParam() {
    return destValueVector;
   }

   public Vector getDestParamType() {
    return destParamVector;
   }

   boolean regChkBxStatus, invokeChkBxStatus;

   public void setRegChkBxStatus(boolean status) {
    regChkBxStatus = status;
   }

   public void setInvokeChkBxStatus(boolean status) {
    invokeChkBxStatus = status;
   }

   /**
    *  Generic method to display message on the screen through a
frame
    * @param  message string
    */
   public void displayMessage(String msgString) {
    setLayout(null);
    setSize(250, 250);
    this.initComponents();
    Label l = new Label(msgString);
    add(l);

    add(m_btn_menu);
    add(m_btn_cancel);

    m_btn_menu.addActionListener(this);
    m_btn_cancel.addActionListener(this);
```

```
 addWindowListener (
  new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
    System.exit(0);
   }
  }
 );

 l.setBounds(80, 70, 200, 20);
 m_btn_menu.setBounds(50, 120, 60, 20);
 m_btn_cancel.setBounds(140, 120, 60, 20);
}

static TextField[] paramTextField;
static Label[] paramLabel;
Vector paramNames;

/**
 *  Method for taking parameters from user
 */
private void showParamInputFrame() {
 paramNames = new Vector();
 paramNames = getParams(getWSName(), getMethodName());
 setLayout(null);
 setSize(250, 250);

 paramTextField = new TextField[paramNames.size()];
 paramLabel = new Label[paramNames.size()];

 int i;
 for (i = 0; i < paramNames.size(); i++) {
  paramLabel[i] = new Label("Param" + i);
  add(paramLabel[i]);
  paramLabel[i].setBounds(30, 30 * (i + 1), 90, 20);
  paramTextField[i] = new TextField();
  add(paramTextField[i]);
  paramTextField[i].setText("");
  paramTextField[i].setBounds(120, 30 * (i + 1), 90, 20);
 }
 add(m_btn_ok);
 m_btn_ok.setBounds(50, 30 * (i + 2), 60, 20);
 add(m_btn_cancel);
 m_btn_cancel.setBounds(140, 30 * (i + 2), 60, 20);

 m_btn_ok.addActionListener(this);
 m_btn_cancel.addActionListener(this);

 addWindowListener (
  new WindowAdapter() {
```

```
      public void windowClosing(WindowEvent e) {
       System.exit(0);
      }
     }
    );
    return;
   }


   Vector destParamVector;
   Vector destValueVector;
   static TextField[] destParamTextField;
   static Label[] destParamLabel;
   Dialog inputDialog;

   /**
    *  Method to get destination parameters from the user
    *
    * @param  destWS      destination web service name
    * @param  destMethod  destination method name
    */
   public void showDestParamInputFrame(String destWS, String
 destMethod) {
    inputDialog = new Dialog(this, true);
    destParamVector = new Vector();
    destParamVector = getDestWSParamsFromDir(destWS, destMethod);

    Panel inputPanel = new Panel();
    inputPanel.setLayout(null);

    destParamTextField = new TextField[destParamVector.size()];
    destParamLabel = new Label[destParamVector.size()];

    int i;
    for (i = 0; i < destParamVector.size(); i++) {
     destParamLabel[i] = new Label("Param" + i);
     inputPanel.add(destParamLabel[i]);
     destParamLabel[i].setBounds(30, 30 * (i + 1), 90, 20);
     destParamTextField[i] = new TextField();
     inputPanel.add(destParamTextField[i]);
     destParamTextField[i].setText("");
     destParamTextField[i].setBounds(120, 30 * (i + 1), 90, 20);
    }
    inputPanel.add(m_btn_invoke);
    m_btn_invoke.setBounds(50, 30 * (i + 2), 60, 20);
    inputPanel.add(m_btn_cancel);
    m_btn_cancel.setBounds(140, 30 * (i + 2), 60, 20);
    m_btn_invoke.addActionListener(this);
    m_btn_cancel.addActionListener(this);
```

```
 inputDialog.add(inputPanel);
 inputDialog.setSize(250, 250);
 inputDialog.setVisible(true);

 addWindowListener (
  new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
    System.exit(0);
   }
  }
 );
}

String wName = "", mName = "";

private void setWSName(String wsName) {
 wName = wsName;
}

private void setMethodName(String methodName) {
 mName = methodName;
}

public String getWSName() {
 return (wName);
}

public String getMethodName() {
 return (mName);
}

/**
 *  Shows initial screen
 */
private void init() {
 setLayout(null);
 setSize(250, 250);

 Label l = new Label("Select one operation on a Wrapper:");
 add(l);
 add(reg_chkbx);
 add(invoke_chkbx);
 add(m_btn_proceed);
 add(m_btn_cancel);

 m_btn_proceed.addActionListener(this);
 m_btn_cancel.addActionListener(this);
```

```
 addWindowListener (
  new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
    System.exit(0);
   }
  }
 );

 l.setBounds(20, 30, 200, 20);
 reg_chkbx.setBounds(30, 50, 120, 10);
 invoke_chkbx.setBounds(30, 90, 120, 10);
 m_btn_proceed.setBounds(50, 120, 60, 20);
 m_btn_cancel.setBounds(140, 120, 60, 20);
}


/**
 *  Initial menu
 */
private void initInvokeMenu() {
 setLayout(null);
 add(m_btn_next);
 add(m_btn_cancel);
 add(ws_label);
 if (invokeChkBxStatus == true) {
  add(method_label);
 }
 add(ws_txt);
 if (invokeChkBxStatus == true) {
  add(method_txt);
 }
 m_btn_next.addActionListener(this);
 m_btn_cancel.addActionListener(this);

 addWindowListener (
  new WindowAdapter() {
   public void windowClosing(WindowEvent e) {
    System.exit(0);
   }
  }
 );
 setSize(250, 250);
 ws_label.setBounds(30, 50, 120, 10);
 ws_txt.setBounds(160, 50, 60, 20);
 if (invokeChkBxStatus == true) {
  method_label.setBounds(30, 80, 120, 10);
  method_txt.setBounds(160, 80, 60, 20);
 }
 m_btn_next.setBounds(50, 120, 60, 20);
```

```
  m_btn_cancel.setBounds(140, 120, 60, 20);
 }

 /**
  *  Initialtes components
  */
 void initComponents() {
  ws_txt.setText("");
  method_txt.setText("");
  reg_chkbx.setState(false);
  invoke_chkbx.setState(false);
 }
}
```