jWST

Table of contents

1. General

1.1. Concepts for WS-BusinessActivity framework implementation

1.1.1. Introduction

This is a site of my student research project. I would like to thank my mentor Mas. Sc. Muhammad Farhat Kaleem for his support during the development of this project. Further I thank Dr. Markus Venzke and Boris Gruschko for their help and advices.

1.1.2. Abstract goal

The goal of my project work is developing of concept and strategies for implementation of transactional communication between web services. The content communication should be SOAP-based, containing the required transactional information and payload.

1.1.3. Concrete goal

First of all I split the abstract goal in two major tasks.

- Analyse of requirements and development of suitable concepts
- Building a sample implementation
- - Building a sample application without transactions
  - Implementing WS-Coordination Framework
  - Implementing WS-BusinessActivity Framework

1.1.3.1. Building a sample application

The sample application consist of some web services. I choose 'travaling agency' example for demonstration of transactional features of my work.
The application consist of a client and two web services: a traveling agency (the client), an airline (the webservice) and a hotel (another webservice).
For simplicity I deployed the webservices on one server and the client on another. The hotel and airline reservation systems has been implemented as trivial J2EE applications. The MySQL DB Server is used for data storage. The J2EE EJB encapsulate the data and the business logic.

1.1.3.2. Implementing WS-Coordination Framework

WS-Coordination specification defines entities and their interactions, needed for the

coordinated activity. The Coordinator provides coordination services to applications, that want to participate in a coordinated activity. The Participant is service deployed on the site of business web service. An Initiator is mostly a client, willing a coordinated activity.

1.1.3.3. Implementing WS-BusinessActivity Framework

WS-BusinessActivity Framework defines two protocols that have to be implemented. It leverage the WS-Coordnation Framework by extending it to support business activities and puts some constraints on it.

1.2. History of Changes

RSS

1.2.1. Version 1.7 (9. September 2005)

1.2.1.1. site

- Changed the site layout and updated some content(sza)

1.2.2. Version 1.6 (5. June 2004)

1.2.2.1. paper

- Began writing on paper about the decision engine component(sza, bgr)
- Paper finished(sza)

1.2.2.2. project

- Source code imported in CVS on Sourceforge(sza)
- Project implementation ended(sza)

1.2.2.3. report

- Report finished(sza)

1.2.3. Version 1.5 (1. May 2004)

1.2.3.1. project

- The major functionality implemented.(sza)
- The project title changed(sza)

1.2.3.2. report

- Began to write the report(sza)

1.2.4. Version 1.4 (16. April 2004)

1.2.4.1. project

- Fixed some bugs.(sza)
- Replaced most used constants through the deployment parameters(sza)

1.2.5. Version 1.3 (06. April 2004)

1.2.5.1. project

- Business to Coordination mapping initial prototype.(sza)
- Deployment descriptors updated.(sza)
- Start of testing the framework with 2 webservices.(sza)

1.2.6. Version 1.2 (15. March 2004)

1.2.6.1. project

- Built a WS-BusinessActivity endpoints(sza)

1.2.7. Version 1.1 (07. March 2004)

1.2.7.1. project

- Developed a WS-Coordination implementation.(sza)
- Developed web service sample application.(sza)

1.2.7.2. site

- Extended description about Apache AXIS, WS-Coordination and WS-BusinessActivity(sza)

1.2.8. Version 1.0 (09. Januar 2004)

1.2.8.1. site

- Initial Import(sza)

1.3. Todo List

1.3.1. medium

- **[site]** Fill the site content. # sza

1.4. Site technology

1.4.1. Overview

This site is build with forrest

comming soon

1.5. Links

1.5.1. Overview

This site is build with forrest

comming soon

1.6. Disclaimer

1.6.1. Property

The content of this site is my intellectual property

Full version : comming soon

1.6.2. Links

I'm not responsible for the content of sites I refer to.

Full version : comming soon

1.6.3. Trademarks

Sun, IBM, Apache, JBoss, MySQL, Adobe are trademarks.

Full version : comming soon

2. Documentation

2.1. Documentation

2.1.1. Overview

This page is a collection of the whole documentation, articles, thoughts etc. to the subject of my project work. As my project work is using different technologies there are three themes and the fourth, the project itself.

Here they are:

- AXIS
- WS-Coordination
- WS-BusinessActivities
- Project

2.2. AXIS

2.2.1. AXIS Overview

2.2.1.1. Introduction

Note:
The following descriptions are partly taken from Apache Axis site. (http://ws.apache.org/axis/)

Axis is essentially a SOAP engine - a framework for constructing SOAP processors such as clients, servers, gateways, etc. The current version of Axis is written in Java, but a C++ implementation of the client side of Axis is being developed. It was developed by IBM and later treaten up on Apache Group for further development.

2.2.1.2. AXIS Key Features

- Speed. Axis uses SAX (event-based) parsing to acheive significantly greater speed than earlier versions of Apache SOAP.
- Flexibility. The Axis architecture gives the developer complete freedom to insert extensions into the engine for custom header processing, system management, or anything else you can imagine.
- Stability. Axis defines a set of published interfaces which change relatively slowly compared to the rest of Axis.

- Component-oriented deployment. You can easily define reusable networks of Handlers to implement common patterns of processing for your applications, or to distribute to partners.
- Transport framework. We have a clean and simple abstraction for designing transports (i.e., senders and listeners for SOAP over various protocols such as SMTP, FTP, message-oriented middleware, etc), and the core of the engine is completely transport-independent.
- WSDL support. Axis supports the Web Service Description Language, version 1.1, which allows you to easily build stubs to access remote services, and also to automatically export machine-readable descriptions of your deployed services from Axis.

2.2.1.3. Links

- http://ws.apache.org/axis/ - the main site of Axis

2.2.2. Apache AXIS Technology

2.2.2.1. Introduction

comming soon

2.2.2.2. Architecture

Put simply, Axis is all about processing Messages. When the central Axis processing logic runs, a series of Handlers are each invoked in order. The object which is passed to each Handler invocation is a MessageContext. A MessageContext is a structure which contains several important parts:

- a "request" message
- a "response" message
- a bag of properties.

Such message path is shown below:

Axis architecture

2.2.2.3. Concepts

Handlers and Chains

AXIS allows the developer of applications to write own Handlers. The composition of Handlers, called Chain is introduced to simplify the management of Handlers.

comming soon

2.2.2.4. Applications

comming soon

2.2.2.5. Links

- Axis architectual guide

2.2.3. Apache AXIS Experience

2.2.3.1. First steps

Installation

During the project definition a question about a convenient Web-Service toolkit occured. The decision has been met to use Apache Axis. In the time of project implementation Apache Axis 1.1 has reached a stable state. Axis 1.2 was avaliable as a Beta-Release from Apache CVS. The installation of both versions were trivial. It is detailed described in the readme-File.

Running samples

The most Apache Axis samples, delivered together with the distributeion were running as written in the readme. Some of them didn't work at all. (e.g. Proxy-Sample)

2.2.3.2. Writing own applications

Web services

The project implementations needs several Web Services to be built. First, the sample scenario Web Services has been implemented using Axis 1.1. This was a straight-forward implementation. The usage of Axis for the Web Service endpoints described in the WS-C and WS-BA specifications were not convenient using the Axis built-in serializer/deserializer.

Due to the modular structure of Apache Axis I used Castor 0.956 for the XML-to-Java mapping as described in the article on developerWorks (see Links). Some difficulties occured during the setup of this symbiose, so the Apache Axis 1.2 has been obtained from the apache CVS and rebuild with Clover as de-/serializer.

Note:

During the build we noticed, that there were some problems with having Clover on your Classpath. If you did, Clover has been tightly bound in Axis libraries and was needed for runtime execution.

Handler

For more excited control on the message processing it is usefull to implement own handlers. There are samples in AXIS documentation describing how to do it.

> **Note:**
> The usage of handlers had the following effect of running in JBoss: the dynamic (in-runtime) undeployment or redeployment of handler in JBoss is impossible and makes the JBoss restarts neccessary. The re-/undeployment works in standalone Tomcat.

2.2.3.3. Problems

The main problem of Axis usage was the leak of sufficient documentation at the time of the project creation. The only usefull documentation sources could be found on IBM and Apache AXIS homepages.

2.2.3.4. Links

- Apache Axis Installation Guide
- Apache Axis main page
- Create Web services using Apache Axis and Castor

2.3. WS-C

2.3.1. WS-Coordination Overview

2.3.1.1. Abstract

> **Note:**
> This content is based on Arjuna Technology Report "Introducing WS-Coordination" (see Resources)

In general terms, coordination is the act of one entity (the known as the coordinator) disseminating information to a number of participants for some domain-specific reason. This reason could be in order to reach consensus on a decision like in a distributed transaction protocol, or simply to guarantee that all participants obtain a specific message, as occurs in a reliable multicast environment. When parties are being coordinated, information known as the coordination context is propagated to tie together operations which are logically part of the same coordinated work or activity.

This context information may flow with normal application messages, or may be an explicit part of a message exchange and is specific to the type of coordination being performed. For

example a security coordination service will propagate differently formed contexts than a transaction coordinator. The fundamental idea underpinning WS-Coordination is that there is indeed a generic need for propagating context information in a Web services environment which is a shared requirement irrespective of the applications being executed. The WS-Coordination specification defines a framework that allows different coordination protocols to be plugged-in to coordinate work between clients, services and participants. Whatever coordination protocol is used, and in whatever domain it is deployed, the same generic requirements are present:

- Instantiation (or activation) of a new coordinator for the specific coordination protocol, for a particular application instance
- Registration of participants with the coordinator, such that they will receive that coordinator's protocol messages during (some part of) the application's lifetime
- Propagation of contextual information between Web services that comprise the application
- An entity to drive the coordination protocol through to completion

2.3.1.2. Activation

The WS-Coordination framework exposes an Activation Service which supports the creation of coordinators for specific protocols and their associated contexts. The process of invoking an activation service is done asynchronously, and so the specification defines both the interface of the activation service itself, and that of the invoking service, so that the activation service can call back to deliver the results of the activation - namely a context that identifies the protocol type and coordinator location.

```
<!-- Activation Service portType Declaration -->
<wsdl:portType name="ActivationCoordinatorPortType">
        <wsdl:operation name="CreateCoordinationContext">
                <wsdl:input message="wscoor:CreateCoordinationContext"/>
        </wsdl:operation>
</wsdl:portType>

<!-- Activation Requester portType Declaration -->
<wsdl:portType name="ActivationRequesterPortType">
        <wsdl:operation name="CreateCoordinationContextResponse">
                <wsdl:input
message="wscoor:CreateCoordinationContextResponse"/>
        </wsdl:operation>
        <wsdl:operation name="Error">
                <wsdl:input message="wscoor:Error"/>
        </wsdl:operation>
</wsdl:portType>
```

2.3.1.3. Registration

Once a coordinator has been instantiated and a corresponding context created by the activation service, a Registration Service is created and exposed. This service allows participants to register to receive protocol messages associated with a particular coordinator. Like the activation service, the registration service assumes asynchronous communication and so specifies WSDL for both registration service and registration requester

```
<!-- Registration Service portType Declaration -->
<wsdl:portType name="RegistrationCoordinatorPortType">
        <wsdl:operation name="Register">
                <wsdl:input message="wscoor:Register"/>
        </wsdl:operation>
</wsdl:portType>

<!-- Registration Requester portType Declaration -->
<wsdl:portType name="RegistrationRequesterPortType">
        <wsdl:operation name="RegisterResponse">
                <wsdl:input message="wscoor:RegisterResponse"/>
        </wsdl:operation>
        <wsdl:operation name="Error">
                <wsdl:input message="wscoor:Error"/>
        </wsdl:operation>
</wsdl:portType>
```

When a participant is registered with a coordinator through the registration service, it receives messages that the coordinator sends (for example, 'completed' and 'compensate' messages if a BusinessAgreement protocol is used); where the coordinator's protocol supports it, participants can also send messages back to the coordinator.

2.3.1.4. Completion

The role of terminator is generally played by the client application, which at an appropriate point will ask the coordinator to perform its particular coordination function with any registered participants - to drive to protocol through to its completion. On completion, the client application may be informed of an outcome for the activity which may vary from simple succeeded-failed notification through to complex structured data detailing the activity's status.

2.3.1.5. Context

The context is critical to coordination since it contains the information necessary for services to participate in the protocol. It provides the glue to bind all of the application's constituent Web services together into a single coordinated application whole. Since WS-Coordination is a generic coordination framework, contexts have to be tailored to meet the needs of specific coordination protocols that are plugged into the framework. The format of a WS-Coordination context is specifically designed to be third-party extensible and its contents

are as follows:

- A coordination identifier with guaranteed global uniqueness for an individual coordinator in the form of a URI
- An address of a registration service endpoint where parties receiving a context can register participants into the protocol
- A TTL value which indicates for how long the context should be considered valid
- Extensible protocol-specific information particular to the actual coordination protocol supported by the coordinator

While the first three points are common sense, the fourth is somewhat more interesting. Since WS-Coordination is generic, it is of very little use to applications without augmentation, and this is reflected in the part of the WS-Coordination XML schema for contexts.

```
<xs:complexType name="CoordinationContextType" abstract="false">
        <xs:complexContent>
                <xs:extension base="wsu:ContextType">
                        <xs:sequence>
                                <xs:element name="CoordinationType"
type="xs:anyURI" />
                                <xs:element name="RegistrationService"
type="wsu:PortReferenceType" />
                                <xs:any namespace="##any"
processContents="lax"
                                                minOccurs="0"
maxOccurs="unbounded" />
                        </xs:sequence>
                </xs:extension>
        </xs:complexContent>
</xs:complexType>
```

2.3.1.6. Summary

WS-Coordination looks set to become the adopted standard for activity coordination on the Web. Out of the box, WS-Coordination provides only activity and registration services, and is extended through protocol plug-ins that provide domain-specific coordination facilities. In addition to its generic nature, the WS-Coordination model is also scales efficiently via interposed coordination which allows arbitrary collections of Web services to coordinate their operation in a straightforward and scalable manner. Though WS-Coordination is generically useful, at time of writing only two protocols that leverages WS-Coordination has been made public: WS-AtomicTransactions and WS-BusinessActivities.

2.3.1.7. Resources

- WS-Coordination specification

- Arjuna Technology Report : Introducing WS-Coordination

2.3.2. WS-Coordination Technology

2.3.2.1. Introduction

2.3.2.2. Concepts

2.3.2.3. Applications

2.3.3. WS-Coordination Solutions

2.3.3.1. Task

According to the WS-Coordination specification, the initiator has to create the coordination context by its activation service. Each message send to participant of a business activity includes this coordination context, to identify the activity in a unique way.

2.3.3.2. Message interceptor

There are several ways to allow the creation of the coordination context before the initiator begins its business activity. One way is to let the initiator implement the interfaces defined in the WS-Coordination. This approach is difficult for adopting exisitng systems to the WS-Coordination.

In this framework another way has been chosen. The client (initiator) is extended in that way, that all messages send by it are intercepted. This interception is always active but works like a transparent proxy on the client system. As soon as client sends a **begin** message and recieves a response from the transactional middleware the interceptor becomes opaque. Each message send from the client is intercepted on the client side and send to the proxy service. This call is extended with information needed for the proxy service to relay the request to the target service. This approach guarantees that each message send after the begin message is redirected to the proxy service and than send to the target web service. On the proxy service it is possible to attach the coordination context to the message.

The opaque interceptor on the client side becomes transparent as soon as it recieves a response on the **end** message, send to the transaction middleware, identifying the end of the business activity.

2.3.3.3. Context creation

All messages send after the **begin** message and before the **end** message are redirected to the proxy service and than delivered to the target web service. The responses to the business messages are taking the same way - they arrives on the proxy service and are then delivered to the client.

As soon the coordination middleware recieves a **begin** message it invoke a createCoordinationContext() method on the Activation service. The created context is then returned and stored in the proxy service. The proxy service provides some additional information to the client side interceptor. This information (proxy client id number) is then used by the client interceptor to identify itself. All calls comming from the same client interceptor are extended with the coordination context hold by the proxy service.

2.3.3.4. Registration

The first business message with corresponding coordination context supplied in SOAP-header reaches the participant web service. Another interceptor catches the message on the participant side and tries to find an activity corresponding to the coordination context in the participant coordination service. If no activity can be found, it initialises the Register operation to specified Registration service. (Alternative it can use own Registration service, and import the activity, so the Registration service will contact the Registration service of the initiator and act itself as participant (subordinated coordinator)). The Registration message must contain enough information for the Registration service to determite which activity the registration requester want to participate.

The WS-Coordination specification ommits this information, so the registration message is extended with the coordination context identifier.

During the registration the participant and initiator coordination services exchanges the coordination protocol endpoint references. After this step the coordination conversation is finished. It prepares the communication parties to further interactions providing them enough information for communication along the established logical connection between the coordination protocol endpoints.

2.4. WS-BA

2.4.1. WS-BusinessActivity Overview

2.4.1.1. Transactions

Note:
This content is based on Arjuna Technology Report "Introducing WS-Coordination" (see Resources)

Distributed systems pose reliability problems not frequently encountered in centralized systems. A distributed system consisting of a number of computers connected by a network can be subject to independent failure of any of its components, such as the computers themselves, network links, operating systems, or individual applications. Decentralization allows parts of the system to fail while other parts remain functioning, which leads to the possibility of abnormal behavior of executing applications. Consider the case of a distributed system where the individual computers provide a selection of useful services, which can be utilized by an application. It is natural that an application that uses a collection of these services requires that they behave consistently, even in the presence of failures. A very simple consistency requirement is that of failure atomicity: the application either terminates normally, producing the intended results, or is aborted, producing no results at all. This failure atomicity property is supported by Atomic transactions, which have the following familiar ACID properties:

- Atomicity: The transaction completes successfully (commits) or if it fails (aborts) all of its effects are undone (rolled back)
- Consistency: Transactions produce consistent results and preserve application specific invariants
- Isolation: Intermediate states produced while a transaction is executing are not visible to other transactions. Furthermore transactions appear to execute serially, even if they are actually executed concurrently. This is typically achieved by locking resources for the duration of the transaction so that they cannot be acquired in a conflicting manner by another transaction
- Durability: The effects of a committed transaction are never lost (except by a catastrophic failure)

A transaction can be terminated in two ways: committed or aborted (rolled back). When a transaction is committed, all changes made within it are made durable (forced on to stable storage such as disk). When a transaction is aborted, all changes made during the lifetime of the transaction are undone. In addition it is possible to nest atomic transactions; where the effects of a nested action are provisional upon the commit/abort of the outermost (top-level) atomic transaction.

2.4.1.2. ACID Transactions may be too strong

Traditional transaction processing systems are sufficient to meet requirements if an application function can be represented as a single top-level transaction. However, this is frequently not the case. Top-level transactions are most suitably viewed as short-lived

entities, performing stable state changes to the system; they are less well suited for structuring long-lived application functions that run for minutes, hours, days, or longer. Long-lived top-level transactions may reduce the concurrency in the system to an unacceptable level by holding on to resources (usually by locking) for a long time. Furthermore, if such a transaction aborts much valuable work already performed will be undone. ACID transactions have another criterion, that must be met of the participants. This is a criterion of highly trust. Given that the industry is moving towards loosely-coupled, coarse-grained interaction model supported by Web services, it has become clear that the semantics of traditional ACID transactions are unsuitable for Web scale deployment. Web services-based transactions differ from traditional transactions in that they execute over long periods between parties which share no trust relationship, they require commitments to the transaction to be negotiated at runtime, and isolation levels have to be relaxed.

2.4.1.3. WS-Coordination Foundations

An important aspect of WS-Transaction that differentiates it from traditional transaction protocols is that a synchronous request/response model is not assumed. This model derives from the fact that WS-Transaction is layered upon the WS-Coordination protocol whose own communication patterns are asynchronous by default.

Protocol View

2.4.1.4. Resources

- WS-Coordination specification
- Arjuna Technology Report : Introducing WS-BusinessActivity

2.4.2. WS-BusinessActivity Technology

2.4.2.1. Introduction

comming soon

2.4.2.2. Concepts

comming soon

2.4.2.3. Applications

comming soon

2.4.3. WS-BusinessActivity Solutions

2.4.3.1. Solutions

comming soon

2.5. Project

2.5.1. Project

2.5.1.1. General

The project demonstrate the WS-BA framework functionality. Especially, the adoptaion of existing Web Services for the usage with WS-BA was an important issue for the project developers.

2.5.2. Scenario

2.5.2.1. Sample Scenario Overview

The following scenario has been implemented as a sample application in order to demonstrate the framework functionality.

A sample travel agency has access to a sample airline Web Service and to a sample hotel Web Service. The agency offers a travel to a given destination by plane which lasts over several days, thereby a booking of a hotel is needed.

The scenario uses two domain-specific entities:

Airline Service

An airline service is implemented as a simple J2EE application, exposing its functionality over a Web Service. The business logic is implemented inside a Session EJB with the following semantic.

The AirlineManagerBean offers three methods to the client:

```
public <FlightInfo>Collection getFlights(String cityNameStart, String
cityNameFinish, Date dateOfFlight)
public FlightReservationInfo bookFlight(Integer flightNumber, String
travelAgencyName)
public boolean cancelReservation(FlightReservationInfo reservationInfo)
```

The relation between the AirlineManagerBean and other data types is shown in the following figure:

Sample Airline Logic

The typical invocation looks like:

1. Call getFlights() for given start, destination and date, retrieving the collection of FlightInfo entities
2. Select a flight and call bookFlight() providing a flight number and the travel agency name, retrieving a FlightReservationInfo with a valid seat number or an error (in form of NoSeatAvailableException)
3. If not needed anymore cancel the reservation calling the cancelReservation() and providing the FlightReservation object

The sample application provides some sample data that can be used during the sample runs. This data is stored in the database and can be adjusted as needed. After the call of bookFlight() the number of seats avaliable is decremented by one. The call of cancelReservation() increments the number of seat by one.

Hotel Service

Similiar to airline, the hotel service is implemented as a simple J2EE application, exposing its functionality over a Web Service. The business logic is implemented inside a Session EJB with the following semantic.

The HotelManagerBean offers four methods to the client:

```
public int getAvailiableRooms(String hotelName, Date begin, Date end)
public ReservationInfo bookRoom(String hotelName, String agencyName, Date
begin, Date end)
public boolean cancelReservation(String hotelName, Integer roomNumber, Date
beginDate, Date endDate)
public String getHotelName(String cityName)
```

The relation between the HotelManagerBean and other entities is depicted in the next figure:

Sample Hotel Logic

Typical invocation looks like:

1. Call getHotelName() providing the city name to retrieve the name of the hotel in the city. (For simplicity, the example assumes, that only one hotel exists)
2. Retrieve the number of rooms avaliable in the hotel in a given period of time calling

getAvaliableRooms().
3. Call bookRoom() in order to book a room and provide additionally the name of travel agency
4. If the order should be canceled call cancelReservation()

The sample application provides sample data for experiments. The sample hotel has a given number of rooms which can be booked. The startDate-endDate interval is evaluated in order to provide the number of avaliable rooms. A particular room is free if no reservation with overlapping interval is present in the system.

2.5.3. Installation Guide

2.5.3.1. Before you begin

In order to install the jWST framework basic knowlegde of application server administration is needed.

2.5.3.2. Needed software and infrastructure

jWST Framework consists of two parts, the middleware component and the participant component. These components should be installed in different application servers. In the scope of project development two JBoss 3.2.3 Application Servers has been used. If running on the same machine the port numbers should be adjusted, as described in the installation notes found on the JBoss homepage. Two sucessfully running JBoss instances is a prerequisite for running the samples.

The project makes use of Apache Axis 1.2 which should be installed on both application servers.

2.5.3.3. Detailed instructions...

3. Downloads

3.1. Resources

3.1.1. Resources

Resources published and developed during the project work are avaliable here.

- Articles
- Presentations
- Additional resources

- JavaDOC API

3.2. Resources : Whitepapers and articles

3.2.1. Whitepapers and articles

Name	Description
unpublished article	This article was sent to the ICSOC04 2004, 2nd International Conference on Service Oriented Computing, New York City, NY USA, but has been rejected by the conference commitee.
Report	This report has been created in the scope of my student work. It is avaliable in german only.

3.3. Resources : Presentations

3.3.1. Presentations

Title	Date	Size	a
Project Overview	03.12.2003	141.312 bytes	pre031203.ppt
Intermediate Presentation	05.04.2004	1.625.600 bytes	pre040405.ppt
Final Presentation	30.06.2004	181.248 bytes	pre040630.ppt

Note:
Microsoft Powerpoint is needed for viewing the presentations.

3.4. Resources : Additional resources

3.4.1. XML-Schemas and WSDLs of the sample application

Name	Description
airline.xsd	Sample Airline entities
airline.wsdl	Sample Airline service description
hotel.xsd	Sample Hotel entities
hotel.wsdl	Sample Hotel service description

3.4.2. XML-Schemas and WSDLs of the middleware web service

Name	Description
transaction.xsd	Transactional Middleware entities
transaction.wsdl	Transactional Middleware service description

3.5. Resources : Project API JavaDoc

3.5.1. Overview

Title	Version	Size	Link
jwst-participant	1.0	61.205 bytes	jwst-participant-doc-1_0.zip