Advanced Issues & Future Trends in WS

Ioannis G. Baltopoulos

Department of Computer Science
Imperial College London

Inverted CERN School of Computing, 2005
Geneva, Switzerland

UDDI4J Overview

- The programmatic interface to a registry is through a set of SOAP messages defined in the UDDI specification.
- The IBM UDDI4J is an open source Java implementation of the UDDI protocol; high level API layered on top of SOAP that enables programmatic access to registries.
- It can be used to
  - search for information on a registry,
  - publish new information to a registry and
  - delete information from a registry.

UDDI4J Basics

Package Breakdown

Structured into a number of packages under `org.uddi4j`:

Packages and contents

Name	**Contents**
`org.uddi4j.client`	contains the client class `UDDIProxy`
`org.uddi4j.datatype`	represents UDDI data objects
`org.uddi4j.request`	contains messages sent to the server
`org.uddi4j.response`	response messages from a UDDI server
`org.uddi4j.transport`	support for pluggable transports
`org.uddi4j.util`	utility classes for various tasks

Accessing the Registry

The most important class in the UDDI4J package is the `org.uddi4j.client.UDDIProxy`. Contains methods to:

- connect to a registry,
- query the registry,
- and process the result.

Creating a Registy Proxy

```
private UDDIProxy proxy;
private void setupProxy(){
    proxy = new UDDIProxy();
    try {
        proxy.setInquiryURL(inquiryURL);
    } catch (MalformedURLException e) {
      // Couldn't create the proxy..
    }
}
```

Locating a technical model

The `find_tModel()` method

The UDDIProxy class defines a `find_tModel()` method for locating technical models by

- name
- categories
- identifiers
- any combination of the above

Using the `find_tModel()` method

```
public TModelList find_tModel(
    String name, CategoryBag c, IdentifierBag I,
    FindQualifiers f, int maxRows)
// Example invocation on a UDDIProxy
proxy.find_tModel(name, null, null, null, 5);
```

Locating a BusinessService

The find_service() method

The UDDIProxy class defines a find_service() method for locating technical models by

- Unique ID (UUID)
- name of the service
- category information of the service
- tModel information of the service
- any combination of the above

Using the find_service() method

```
public ServiceList find_service(
    String businessKey, Vector names, CategoryBag c,
    TModelBag t, FindQualifiers f , int maxRows)
```

Locating a BusinessEntity

The find_business() method

The UDDIProxy class defines a find_business() method for locating technical models by

- name of the business
- discoveryURL
- identifier of the business
- category of the business
- tModel information of the service
- any combination of the above

Using the find_business() method

```
public BusinessList find_business(
    Vector names, DiscoveryURLs d, IdentifierBag i,
    CategoryBag c, TModelBag t, FindQualifiers f,
    int maxRows)
```

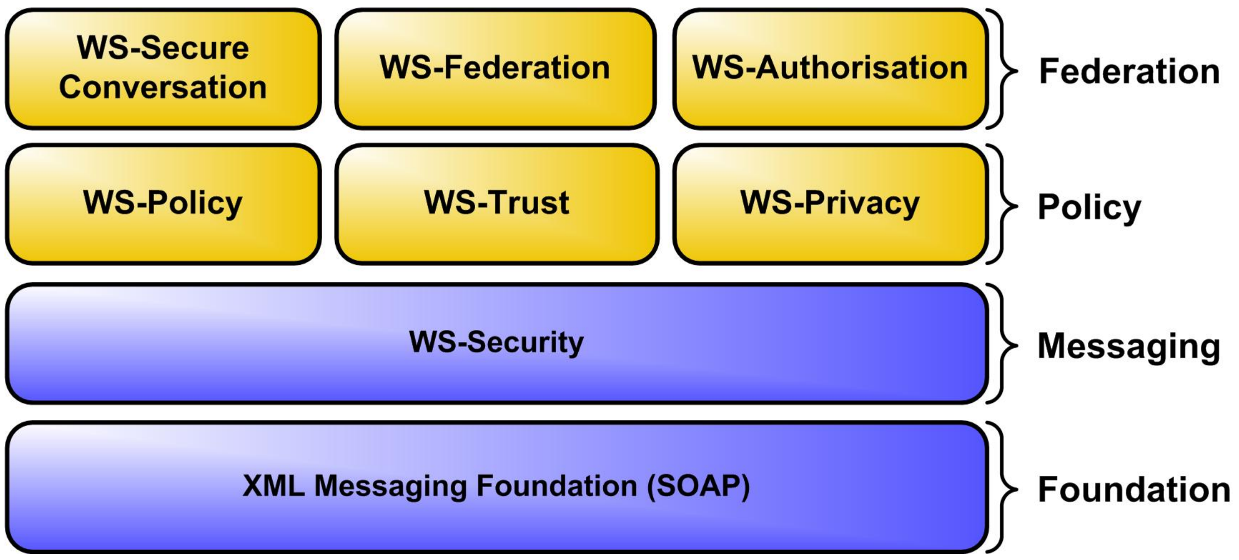
Security Requirements

- **Confidentiality**
  Ensures that only authorised parties access the information.
- **Authentication**
  Ensures the originator of a message can provide appropriate proof of identity.
- **Integrity**
  Ensures that a message isn't modified accidentally or intentionally in transit.
- **Nonrepudiation**
  Guarantees that neither sender or receiver of a message can deny its transmission.
- **Authorization**
  Ensures that entities with given identity are given access to resources.

WS-Security
The Web Services Security Roadmap

- The Web services security roadmap laid out by IBM and Microsoft is composed of a whole suite of specifications covering various facets of security (messaging, policies, trust, privacy, etc.).
- The specifications build upon one another and are all built on top of a single specification, WS-Security, that defines a message security model.
- Currently the model for securing Web services consists of 7 specifications.

WS-Security Roadmap



WS-ReliableMessaging
Motivating the Solution

Some problems

The current implementation of Web Services lacks guarantees of

- Message Ordering
- Once and only once delivery
- Network/Machine availability

The solution!

A standard (therefore interoperable way) that would take care of all the above problems at the middleware layer.
IBM, Microsoft, TIBCO and BEA are working together to develop a SOAP extension model to help solve these types of problems, and the result is WS-ReliableMessaging.

WS-RM Processing Model

1. A client application sends a new message to the SOAP client.
2. The SOAP client, using WS-RM code, associates a unique identifier for this message and saves it in a persistent store.
3. The WS-RM client tries to send the message to the target server. If it fails it retries until it times-out.
4. Upon receiving the message, the WS-RM server code acknowledges receipt by sending an acknowledgment header.
5. After receiving the acknowledgment, the WS-RM client removes the message and the state information from the persistent store.
6. The SOAP server locates and invokes the desired Web Service.
7. Once the service is invoked, the message can be sagely removed from the WS-RM sever-side runtime persistent store.
8. After the Expiration time has passed, the WS-RM server runtime can remove the state information about the particular message sequence.

WS-Coordination

Introducing transactions to Web Services

Definition

A transaction is the scope under which a unit of work is defined. The size or breadth of the amount of work will vary between applications.

- Intuitively, the above definitions means considering several successive calls as a single atomic one.
- This is particularly useful for Banking applications or Business systems where several subsystems need to be updated and either `all` or `none` of the updates succeed.

Concluding Remarks

In this lecture we saw

- A programmatic interface to the UDDI Registry using IBM's open source UDDI4J
- The Web Services Security Roadmap (WS-Security)
- Current work in transactions and reliable messaging
- Finally, future uses on the Grid

Thank you!