Web Services Foundations (2): SOAP, WSDL and UDDI

Helen Paik

School of Computer Science and Engineering
University of New South Wales

References used for the Lecture:

- Webber Book Chapter 7
- Blue Book Chapters 5-7
- Mike Book Chapter 6
- http://www.soaspecs.com/ws.php

Acknowledgement: Some other materials is adapted from COMP9322 lectures in previous sessions, which are prepared by Dr. Helen H-Y Paik, Dr. Sherif Sakr.

Week 3

Part I

More on SOAP/WSDL

Designing WS Interface with SOAP/WSDL

Given an operation:

```
Operation name: concat
Parameters: (st1: string, st2: string)
Return: string
```

More precisely as WS operation ...

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: st1
  Type: string in http://www.w3.org/2001/XMLSchema
Part 2:
  Name: st2
  Type: string in http://www.w3.org/2001/XMLSchema
Output message:
Part 1:
   Name: return
   Type: string in http://www.w3.org/2001/XMLSchema
```

Designing WS Interface with SOAP/WSDL - RPC style

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: st1
  Type: string in http://www.w3.org/2001/XMLSchema
Part 2:
  Name: st2
  Type: string in http://www.w3.org/2001/XMLSchema
Output message:
Part 1:
   Name: return
   Type: string in http://www.w3.org/2001/XMLSchema
```

```
<ese:concat xmlns:ese="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
    <st1 xsi:type="xsd:string">service</st1>
    <st2 xsi:type="xsd:string">foundry</st2>
</ese:concat>
```

```
<ese:concatResponse xmlns:ese ="http://sltf.unsw.edu.au/eg"
                    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                    xmlns:xsi="http://www.w3.org/2001/XMLSchema-Instance">
   <return xsi:type="xsd:string">service foundry</return>
</ese:concatResponse>
```

Designing WS Interface w/ SOAP/WSDL: Document-style

(note - We first should define the message types)

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Type of concatRequest defined as a schema

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: concatRequest
  Element: concatRequest in http://sltf.unsw.edu.au/eg
Output message:
...
```

Designing WS Interface w/ SOAP/WSDL: Document-style

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

Type of concatResponse also defined as a schema

```
Local name: concat
Namespace: http://sltf.unsw.edu.au/eg
Input message:
Part 1:
  Name: concatRequest
  Element: concatRequest in http://sltf.unsw.edu.au/eg
Output message:
Part 1:
  Name: concatResponse
  Element: concatResponse in http://sltf.unsw.edu.au/eg
```

Designing WS Interface w/ SOAP/WSDL: Document-style

```
<xsd:schema targetNamespace="http://sltf.unsw.edu.au/eg"
            xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <xsd:element name="concatRequest">
      <xsd:complexType>
         <xsd:sequence>
            <xsd:element name="st1" type="xsd:string"/>
            <xsd:element name="st2" type="xsd:string"/>
         </xsd:sequence>
      </xsd:complexType>
   </xsd:element>
   <xsd:element name="concatResponse" type="xsd:string"/>
</xsd:schema>
```

These SOAP messages can be validated against the schema

```
<ese:concatRequest xmlns:ese="http://sltf.unsw.edu.au/eg">
    <st1>service</st1>
    <st2>foundry</st2>
</ese:concatRequest>
```

```
<ese:concatResponse xmlns:ese ="http://sltf.unsw.edu.au/eg">
   service foundry
</ese:concatResponse>
```

Fault Handling in SOAP

A SOAP `Fault` message is reserved for providing an extensible mechanism for transporting structured and unstructured information about problems that have arisen during the processing of SOAP messages.

Because clients can be written on a variety of platforms using different languages, there must exist a standard, platform-independent mechanism for communicating the error. SOAP provides a platform-independent way of describing the error within the SOAP message using a SOAP fault.

- `Fault` element is located inside body of the message
- Two mandatory elements: `Code` and `Reason`
- `Code` - e.g., `VersionMismatch`, `MustUnderStand`, `DataEncodingUnknown`, `Sender`
- `Reason` - Human readable description of the fault
- and other elements (see documents: JAX-WS SOAP Faults)

SOAP faults appear in the SOAP body section

e.g., SOAP v1.2

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope" ...
<env:Body>
   <env:Fault>
      <env:Code>
         <env:Value>env:Sender</env:Value>
         <env:Subcode>
            <env:Value>rpc:BadArguments</env:Value>
         </env:Subcode>
      </env:Code>
      <env:Reason>
           <env:Text xml:lang="en-US">Processing error</env:Text>
      </env:Reason>
      <env:Detail>
           <e:myFaultDetails xmlns:e="http://travelcompany.example.org/faults">
              <e:message>Name does not match card number</e:message>
              <e:errorcode>098</e:errorcode>
           </e:myFaultDetails>
      </env:Detail>
   </env:Fault>
 </env:Body>
</env:Envelope>
```

SOAP faults appear in the SOAP body section

e.g., SOAP v1.1 (simpler structure)

```
<?xml version="1.0"?>
<soap:Envelope
    xmlns:soap='http://schemas.xmlsoap.org/soap/envelope'>
   <soap:Body>
      <soap:Fault>
         <faultcode>soap:VersionMismatch</faultcode>
         <faultstring, xml:lang='en">
            Message was not SOAP 1.1 compliant
         </faultstring>
         <faultactor>
            http://sample.org.ocm/jws/authnticator
         </faultactor>
      </soap:Fault>
   </soap:Body>
</soap:Envelope>
```

Modelled SOAP faults - in WSDL definition (example from Oracle Docs)

```
<definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://examples/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns="http://schemas.xmlsoap…
   targetNamespace="http://examples/" name="HelloWorldService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://examples/" schemaLocation="http://localhost:…/>
    </xsd:schema>
  </types>
  <message name="sayHelloWorld">
    <part name="parameters" element="tns:sayHelloWorld" />
  </message>
  <message name="sayHelloWorldResponse">
    <part name="parameters" element="tns:sayHelloWorldResponse" />
  </message>
  <message name="MissingName">
    <part name="fault" element="tns:MissingName" />
  </message>
  <portType name="HelloWorld">
    <operation name="sayHelloWorld">
      <input message="tns:sayHelloWorld" />
      <output message="tns:sayHelloWorldResponse" />
      <fault message="tns:MissingName" name="MissingName" />
    </operation>
  </portType>
  <binding name="HelloWorldPortBinding" type="tns:HelloWorld">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
      style="document" />
    <operation name="sayHelloWorld">
      <soap:operation soapAction="" />
      <input>
        <soap:body use="literal" />
      </input>
      <output>
        <soap:body use="literal" />
       </output>
      <fault name="MissingName">
        <soap:fault name="MissingName" use="literal" />
      </fault>
    </operation>
  </binding>
  <service name="HelloWorldService">
    <port name="HelloWorldPort" binding="tns:HelloWorldPortBinding">
      <soap:address
        location="http://localhost:7001/HelloWorld/HelloWorldService" />
    </port>
  </service>
```

Modelled SOAP faults - in service-side code

Web services throws the custom Exception- MissingName

```
package examples;
import javax.jws.WebService;

@WebService(name="HelloWorld", serviceName="HelloWorldService")
public class HelloWorld {
   public String sayHelloWorld(String message) throws MissingName {
        System.out.println("Say Hello World: " + message);
        if (message == null || message.isEmpty()) {
           throw new MissingName();
        }
      return "Here is the message: '" + message + "'";
      }
}
```

Custom Exception (MissingName.java)

```
package examples;
import java.lang.Exception;

public class MissingName extends Exception {
    public MissingName() {
        super("Your name is required.");
    }
}
```

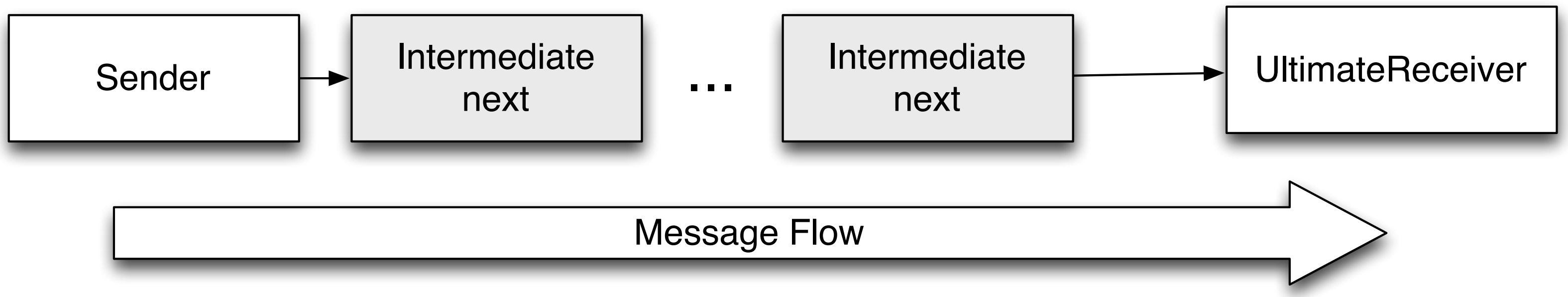
Modelled SOAP faults - in SOAP

The following shows how the SOAP fault is communicated in the resulting SOAP message when the MissingName Java exception is thrown.

```xml
<?xml version = '1.0' encoding = 'UTF-8'?>
<S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
   <S:Body>
      <S:Fault xmlns:ns4="http://www.w3.org/2003/05/soap-envelope">
         <faultcode>S:Server</faultcode>
         <faultstring>Your name is required.</faultstring>
         <detail>
            <ns2:MissingName xmlns:ns2="http://examples/">
               <message>Your name is required.</message>
            </ns2:MissingName>
         </detail>
      </S:Fault>
   </S:Body>
</S:Envelope>
```

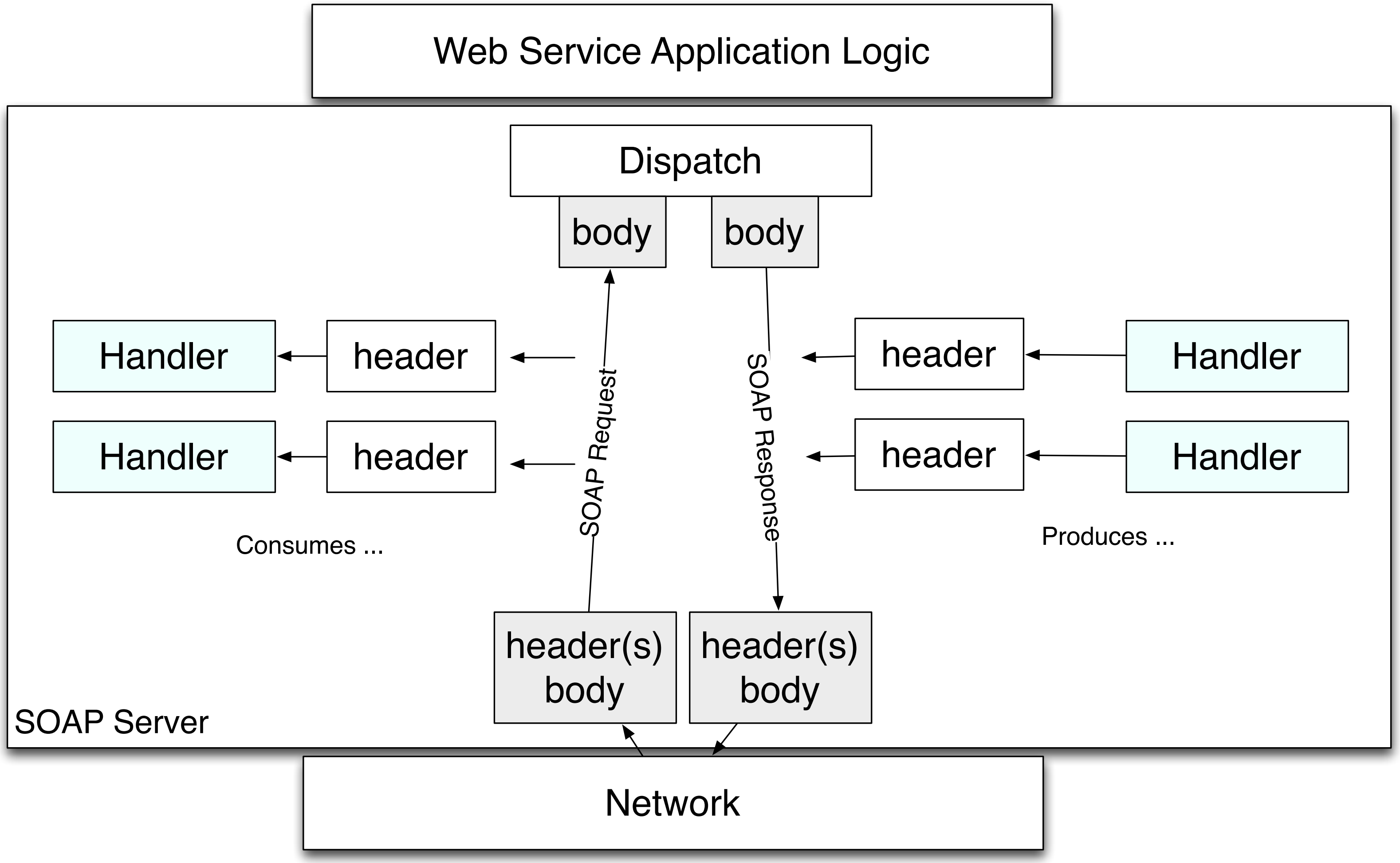
SOAP Processing Model and SOAP Headers

- the lifecycle of a single SOAP message (from the initial sender to the final receiver)
- The messages pass through a number of intermediate *nodes* between the sender and the receiver.
  - Initial Sender: The message originator
  - Ultimate Receiver: The intended recipient
  - Intermediaries: Processing blocks that operate on the soap message before it reaches the ultimate receiver (a SOAP intermediary is a node that acts as both a sender and a receiver at the same time).



SOAP processing model

In a SOAP server (handlers == nodes):



Request: process the header blocks, Response: generates the header blocks

SOAP Processing Model

The intermediaries work by intercepting messages, performing their function, and forwarding the (altered) message to the ultimate receiver.

Common examples of intermediaries would be:

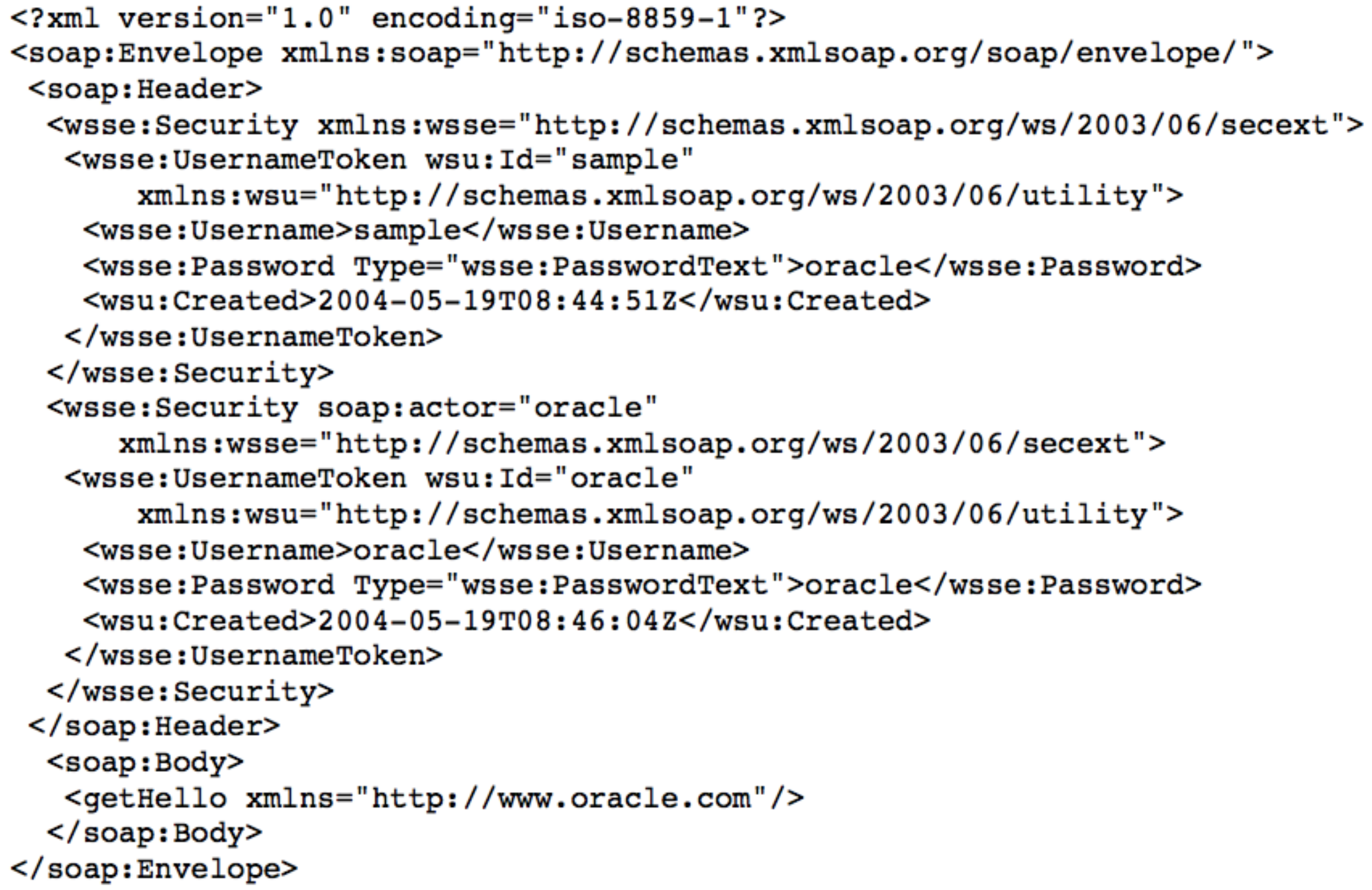
- logging
- encryption/decryption intermediary
- caching ...

Above all, using this model, it is possible to build 'extensions' to basic SOAP (e.g., supporting transaction, different security standards)

A SOAP engine 'implements' a model of their own based on these basics (cf. Axis2 or CXF architecture documentations).

SOAP Headers

header blocks should contain information that influences payload processing (e.g., WS-security standard: a credentials element that helps control access to an operation - docs.oracle.com)



```xml
<?xml version="1.0" encoding="iso-8859-1"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Header>
  <wsse:Security xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="sample"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>sample</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:44:51Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
  <wsse:Security soap:actor="oracle"
      xmlns:wsse="http://schemas.xmlsoap.org/ws/2003/06/secext">
   <wsse:UsernameToken wsu:Id="oracle"
       xmlns:wsu="http://schemas.xmlsoap.org/ws/2003/06/utility">
    <wsse:Username>oracle</wsse:Username>
    <wsse:Password Type="wsse:PasswordText">oracle</wsse:Password>
    <wsu:Created>2004-05-19T08:46:04Z</wsu:Created>
   </wsse:UsernameToken>
  </wsse:Security>
 </soap:Header>
  <soap:Body>
   <getHello xmlns="http://www.oracle.com"/>
  </soap:Body>
</soap:Envelope>
```

Summary

- Binding in WSDL defines (i) message encoding format (ii) transport protocol details. (...) and (...) are two options available in <soap:binding style='...'>.
- A SOAP server employs a pipeline based SOAP message processing model which includes: a sender, ultimate receiver and a series of (...).
- Each (...) is responsible for processing a (...)
- Can you roughly draw a diagram to illustrate how in-bound and out-bound SOAP messages are handled by a SOAP server?
- Why, would you say, is this type of processing model important in using SOAP for WS communication?
- SOAP message body can contain a normal response or a fault through (...)
- Where are the details of faults (if any) by a service declared ?

Part II

UDDI - Advertising/Discovering Services

Service Registries

- To *discover* Web services, a service registry is needed. This requires describing and registering the Web service.
- Publication of a service requires proper description of a Web service in terms of business, service, and technical information.
- Registration deals with persistently storing the Web service descriptions in the Web services registry.
- **Two types of registries can be used**:
  - **The document-based registry**: enables its clients to publish information, by storing XML-based service documents such as business profiles or technical specifications (including WSDL descriptions of the service).
  - **The meta-data-based service registry**: captures the essence of the submitted document.

Service Discovery

- **Service discovery** is the process of locating Web service providers, and retrieving Web services descriptions that have been previously published.
- Interrogating services involve querying the service registry for Web services matching the needs of a service requestor.
  - A query consists of search criteria such as: the type of the desired service, preferred price and maximum number of returned results, and is executed against service information published by service provider.
- After the discovery process is complete, the service developer or client application should know the exact location of a Web service (URI), its capabilities, and how to interface with it.

Types of service discovery

Static:

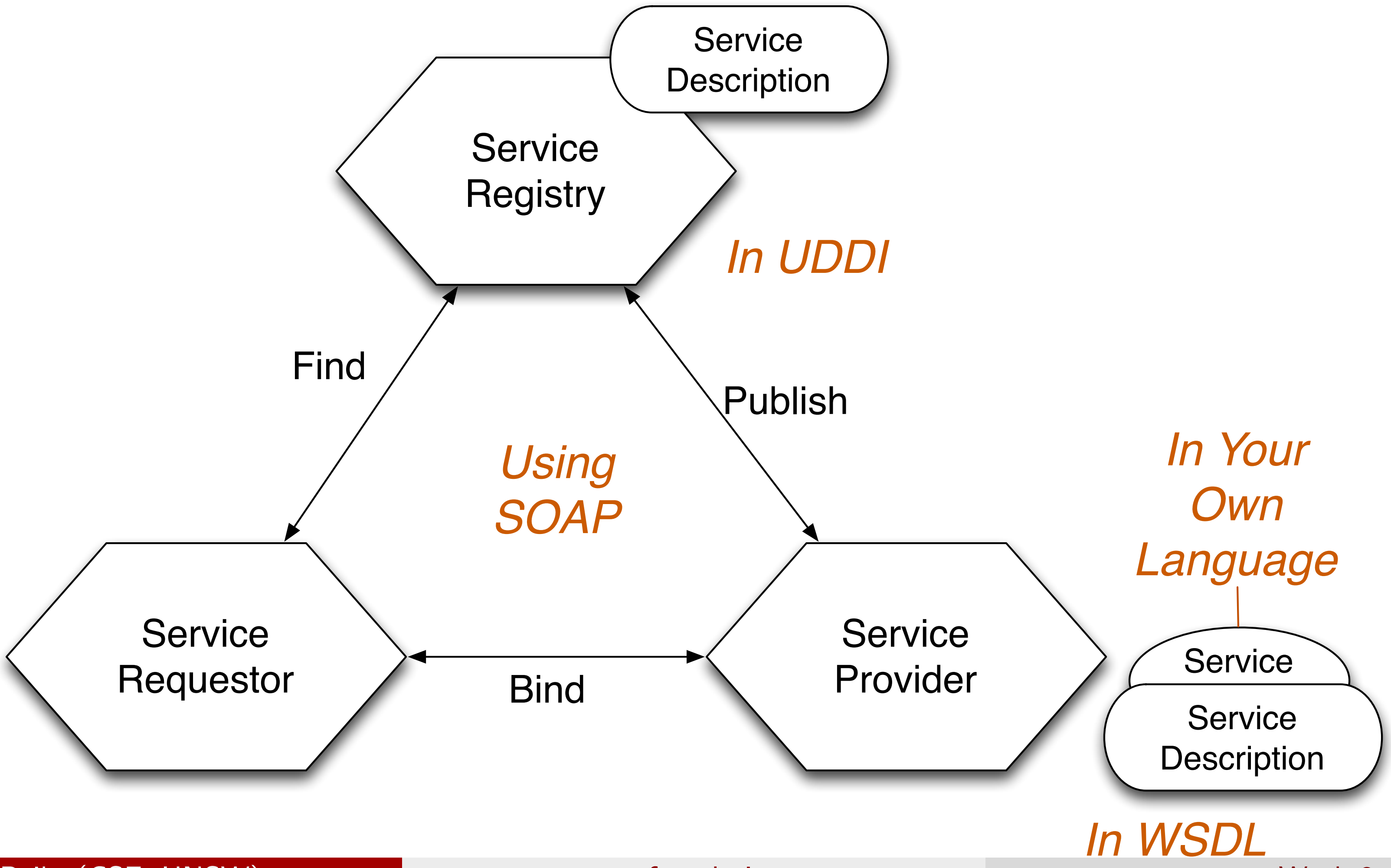
- The service implementation details are bound at design time and a service retrieval is performed on a service registry.
- The results of the retrieval operation are examined usually by a human designer and the service description returned by the retrieval operation is incorporated into the application logic.

Dynamic:

- The service implementation details are left unbound at design time so that they can be determined at run-time.
- The Web service requestor has to specify preferences to enable the application to **infer/reason** which Web service(s) to choose
- Based on application logic quality of service considerations such as best price, performance or security certificates. The application chooses the most appropriate service, binds to it, and invokes it.

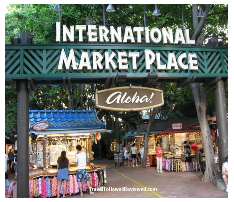
UDDI

Universal Description Discovery and Integration



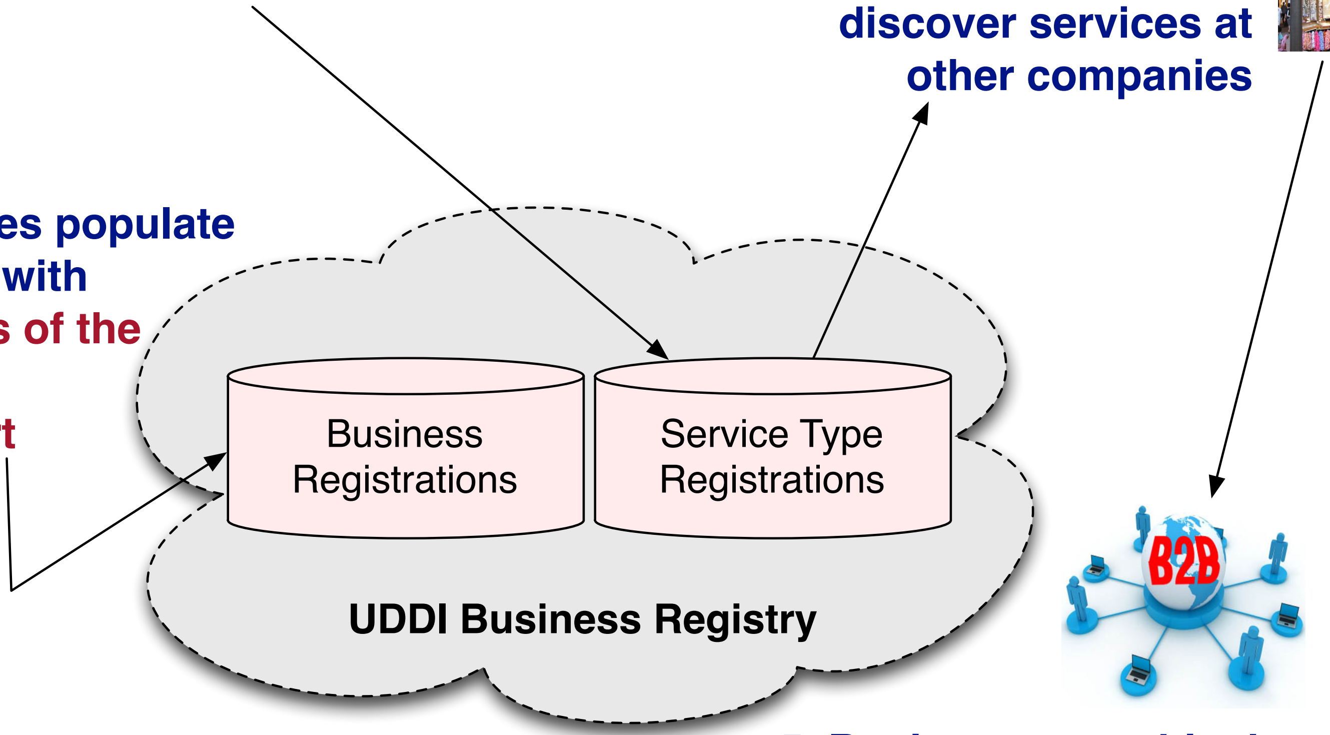
UDDI and the big idea

1. SW companies, standard bodies, and programmers populate the registry with description of various types of services



4. Marketplaces, search engines and business apps query the registry to discover services at other companies

2. Businesses populate the registry with descriptions of the services they support

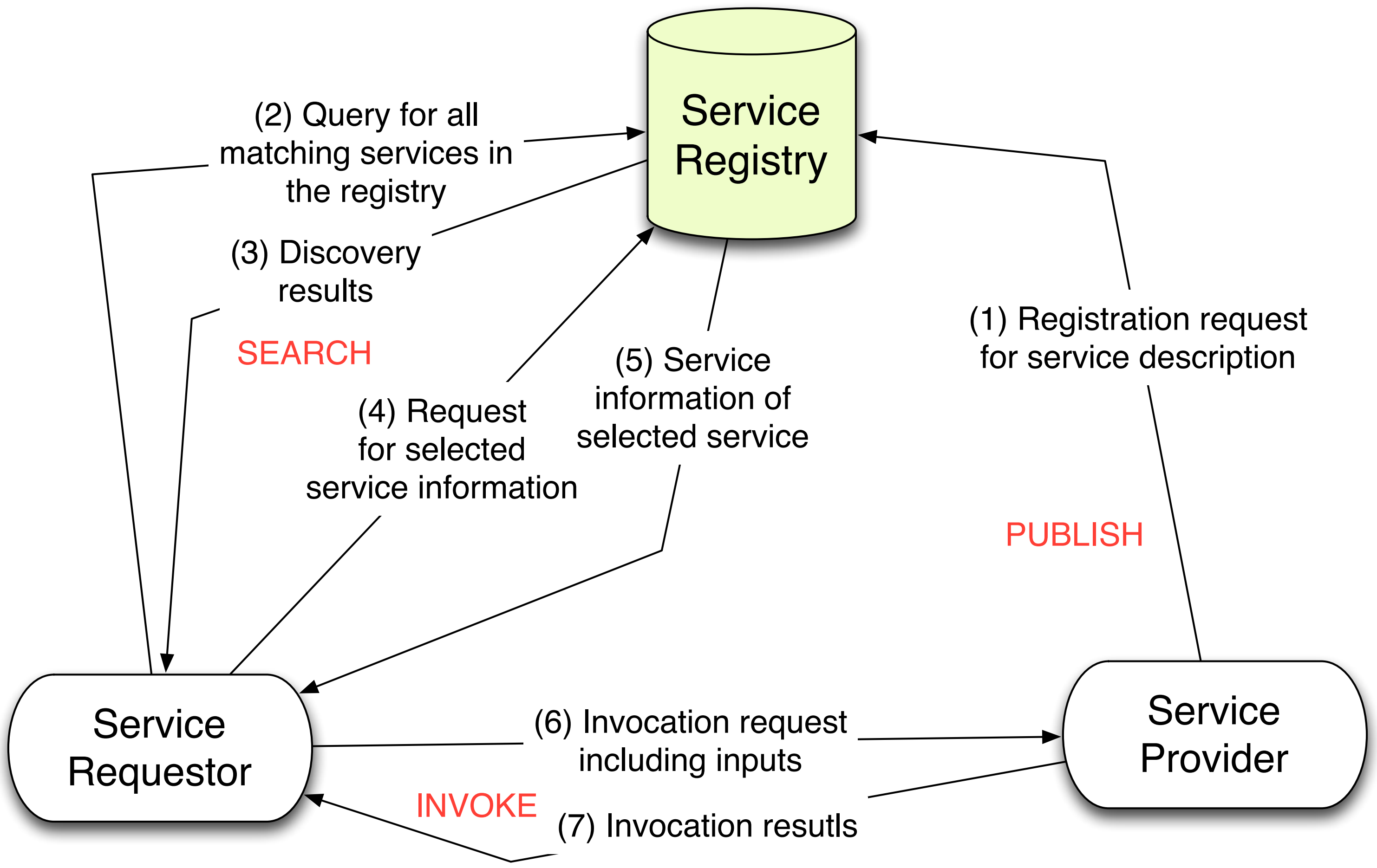


3. UBR assigns a unique identifier to each service and business registration

5. Business uses this data to facilitate easier integration with each other over the web

UDDI and the big idea

How to find the service you want among a potentially large collection of services and servers. The client does not necessarily need to know a priori where the server resides or which server provides the service.


Service Registry
(2) Query for all matching services in the registry
(3) Discovery results
SEARCH
(4) Request for selected service information
(5) Service information of selected service
(1) Registration request for service description
PUBLISH
Service Requestor
Service Provider
(6) Invocation request including inputs
INVOKE
(7) Invocation resutls

UDDI

- UDDI is a registry (not repository) of Web services
- IBM and Microsoft *used* to host public UDDI registry
- Before UDDI, there was no standard way of finding documentation or the location of a particular remote object. Ad-hoc documentation may look like:

```
Contact person: John Smith
COM+ Object: GetWeatherInfo
COMP+ Server: http://bindingpoint.com
Relative URL: /metero/weather
Proxy Location: /Instal/GetWeatherInfo.dll
Description: Returns today's weather. It requires a zip code ...
```

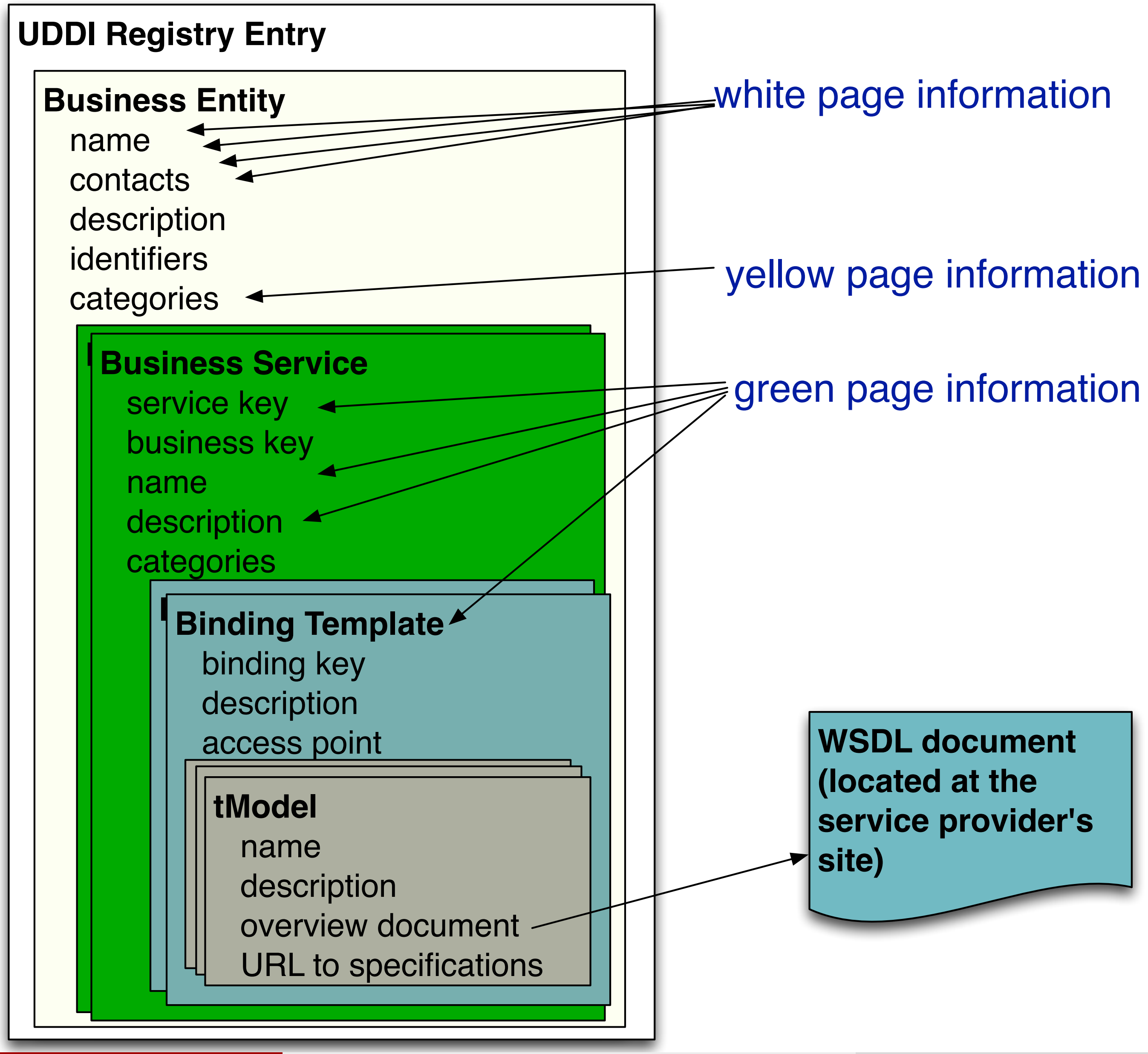
- UDDI is not part of W3C standard (unlike SOAP, WSDL)
- The main standard body for it is OASIS http://www.oasis-open.org, http://uddi.xml.org

UDDI

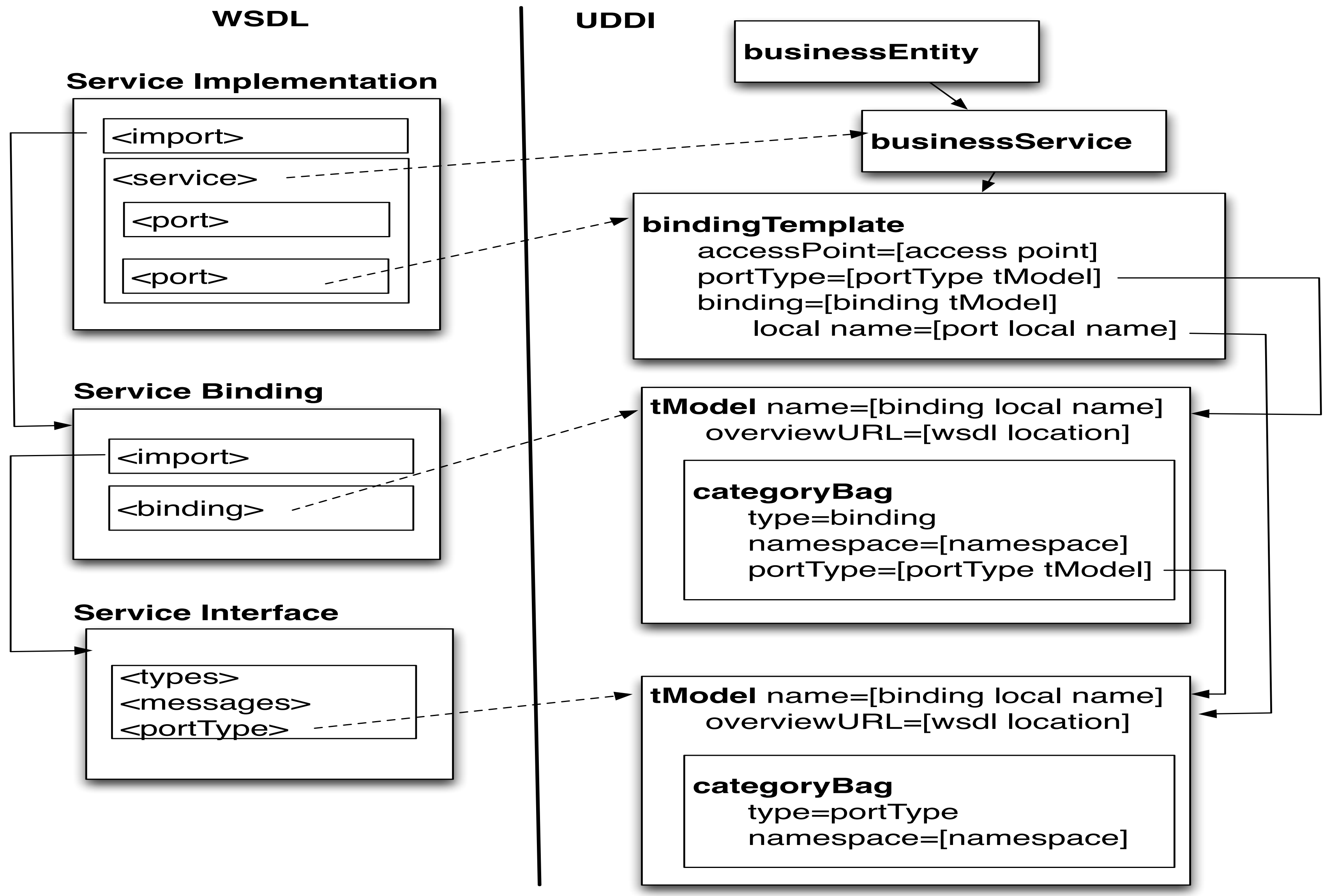
UDDI shares similarities with telephone directories.

- **White Pages**: Contact information about the service provider company. This information includes the business or entity name, address, contact information, other short descriptive information about the service provider, and unique identifier with which to facilitate locating this business
- **Yellow Pages**: Categories under which Web services implementing functionalities within those categories can be found
- **Green Pages**: Technical information about the capabilities and behavioral grouping of Web services

UDDI - overview of its data structure



WSDL and UDDI Mapping (e.g., jUDDI Apache Project)



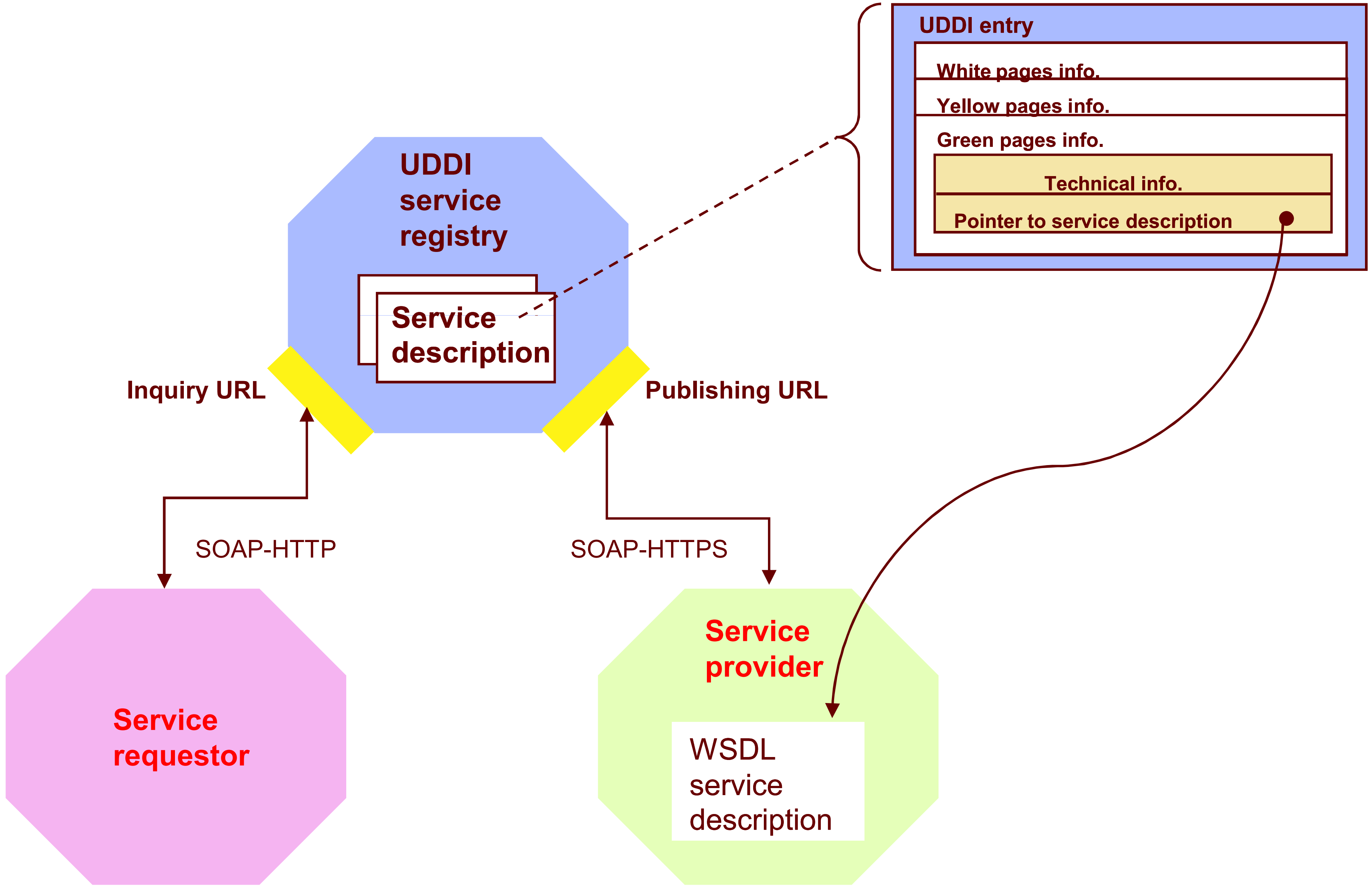
UDDI

- UDDI registry can be browsed by human
- UDDI registry can be programmatically accessed
  - Inquiry API: enable lookup of registry information
  - Publishers API: allow applications to register services
  - an XML schema for SOAP message is defined
  - SOAP is used as the communication protocol
- An example implementation (jUDDI by Apache)
  - http://juddi.apache.org
  - http://juddi.apache.org/docs/3.2/juddi-client-guide/html/

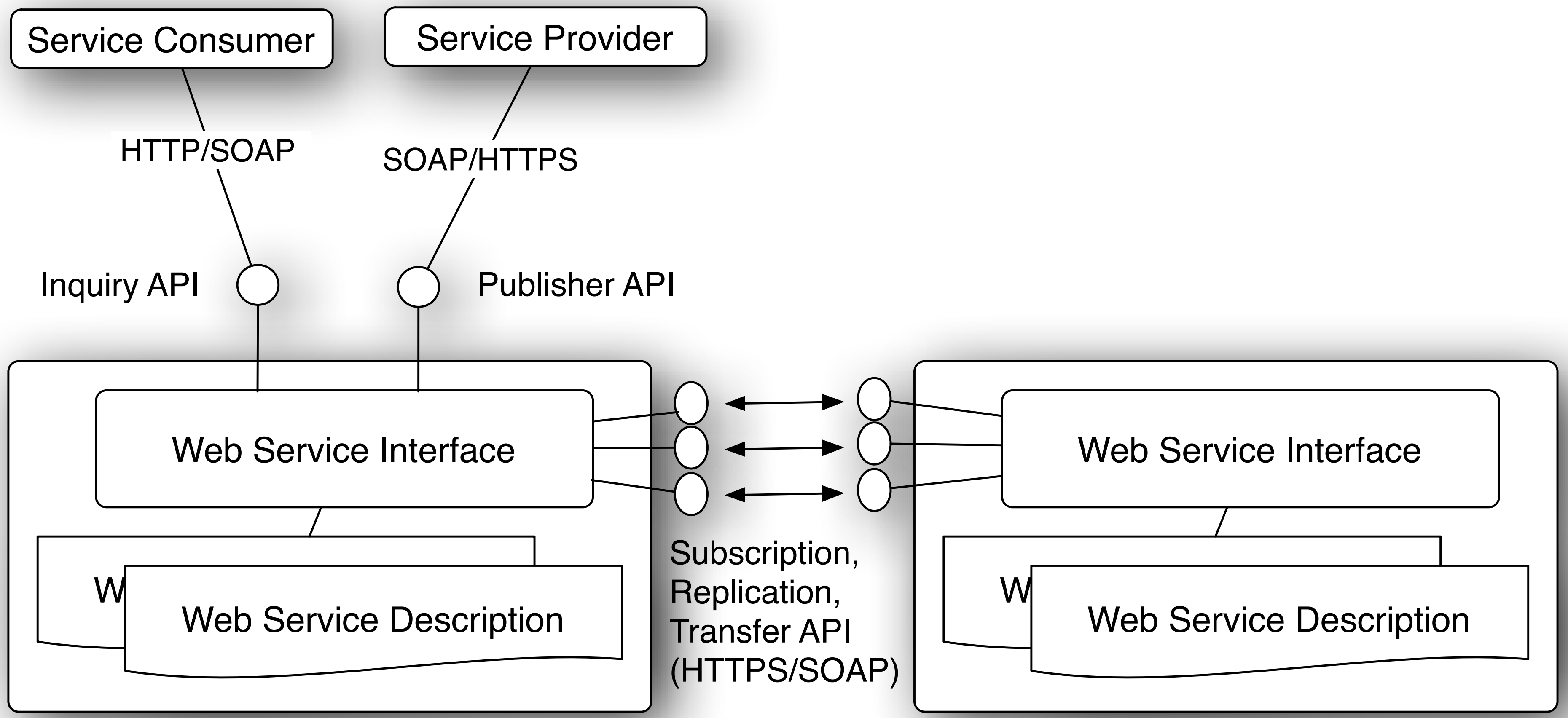
UDDI's provided APIs

- **UDDI provides a SOAP-based API to the business registry.**
- **UDDI Inquiry APIs**: includes operations to find registry entries.
  - Browse, Drilldown, Invocation Patterns
- **UDDI Publish APIs**: operations to add, modify and delete entries in the registry.
- **UDDI Security API**: for access control to the UDDI registry.
- **UDDI Subscription API**: for clients to subscribe to changes of information in the UDDI registry.
- **UDDI Replication API**: to perform replication of information across nodes in a UDDI registry.

UDDI's provided APIs



Interaction with and between UDDIs



How UDDI could play out: an opinion (Webber Book pp.136-141)

Question: Is it reasonable to assume that “people” will search a service registry using APIs to select a service during design time?

- Most likely no ... that’s just not how “people” do business
- You will browse, ask around (word of mouth), google, etc.
- Oftentimes, the selection criteria can be tricky (e.g., existing business relationships, cutting deals, etc.)

If UDDI is not going to be useful in selecting services ... then what?

Note: Most of the UDDI registries in place today are private registries operating inside companies or maintained by a set of companies in a private manner

How UDDI could play out: an opinion (Webber Book pp.136-141)

... UDDI might be useful at runtime ...

Case 1: Service Life-cycle Management

Consider the issues you have to deal with after Web services are deployed and clients are using them

- Overtime, some changes might have to be made (not only the code, but also the physical environment that the service is deployed in)
- e.g., migration to a new server, multiple mirror servers, routine maintenance on the server ...
- Applications that rely on Web services need to stay updated with the latest access end-point information
- How do we propagate the changes to the access point?

UDDI can play the runtime broker/middleman in handling and propagating these changes ...

How UDDI could play out: an opinion (Webber Book pp.136-141)

Scenario: Service Life-cycle management with UDDI

- A Web service is selected for use (searched in or outside UDDI)
- Save (in your local database) the bindingTemplate information of the service from UDDI
- Develop an application using the service
- If the service call fails (or times out):
  - query UDDI for the lastest information
  - compare the info. with the saved info.
  - if different, try calling the service again with the new info.
  - update your local bindingTemplate if needed

How UDDI could play out: an opinion (Webber Book pp.136-141)

... UDDI might be useful at runtime ...

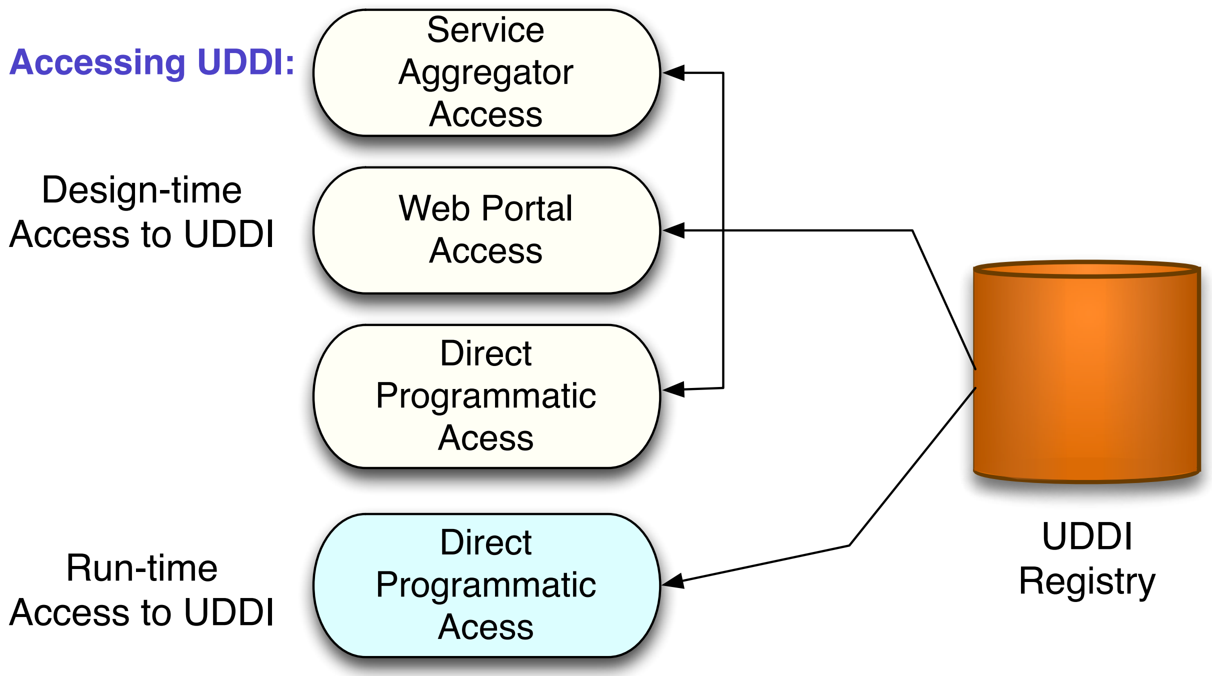
Case 2: Dynamic access point management

Not only when a service call is failed, you may want to dynamically manage and select the most appropriate access point for a service.

- A service may be available from multiple geographical locations
- The client application may have been developed in one country and later used in another county

The concept is similar to downloading files from different mirror sites. The access point can be hardwired in the client application, but by dynamically selecting the most appropriate access point (based on certain criteria) may lead to increased performanace.

How UDDI could play out: an opinion (Webber Book pp.136-141)


Accessing UDDI:
Service Aggregator Access
Design-time Access to UDDI
Web Portal Access
Direct Programmatic Acess
Run-time Access to UDDI
Direct Programmatic Acess
UDDI Registry

Desgin time access – via manual search or direct API acccess, to search and discover services during the application design phase

Runtime access (UDDI playing a brokering/middleman role) – via direct API access, it offers possibilities to build more robust and flexible applications

Static Discovery of Web Services ...

There are some public Web service registries operating (not following UDDI):

- XMETHODS (seems to be offline nowadays): http://www.xmethods.net/ve2/Directory.po
- WebserviceX.NET: http://www.webservicex.net/WS/wscatlist.aspx
- WebServiceList: http://www.webservicelist.com (with user rating info)

Summary

- Can you describe a service registration process and a service discovery process?
- What is the purpose of a WSDL to UDDI mapping model?
- Can you list some of the operations in UDDI API?
- Alternative/suggested use of UDDI ...?

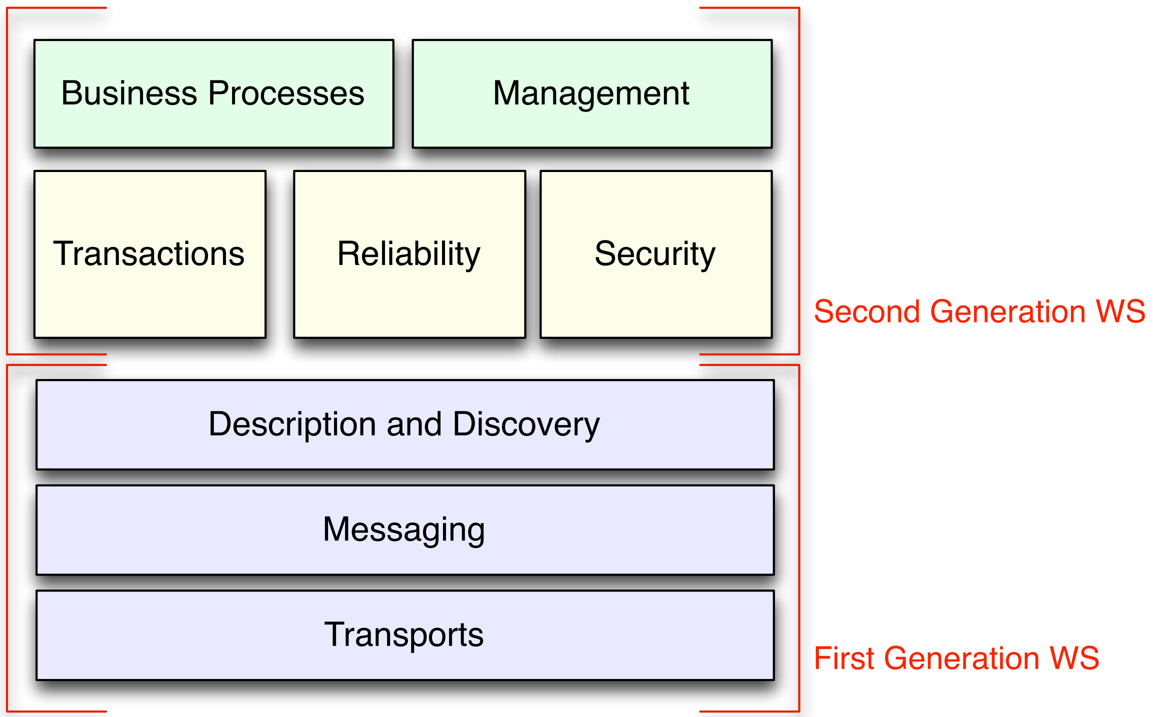
Part III

WS-*: Web Service Extensions

Web Service Standards: WS-* extensions

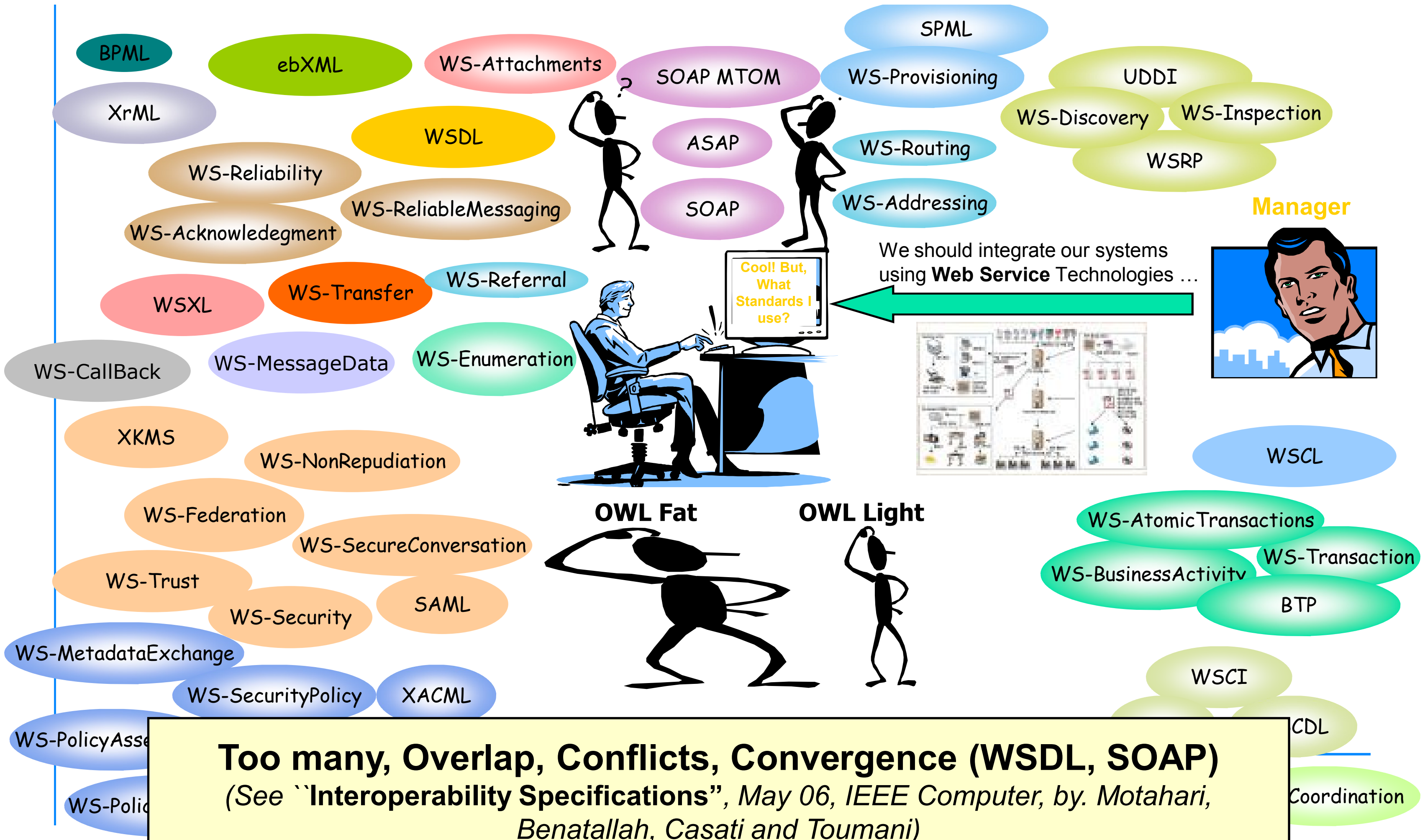
- The term "WS-*" refers to the second generation of Web services standards/specifications.
- On top of the basic standards (WSDL, SOAP and UDDI), these extensions focus on providing supports for various issues in enterprise computing environment

http://www.ibm.com/developerworks/webservices/standards/



Web Service Standards: WS-* extensions

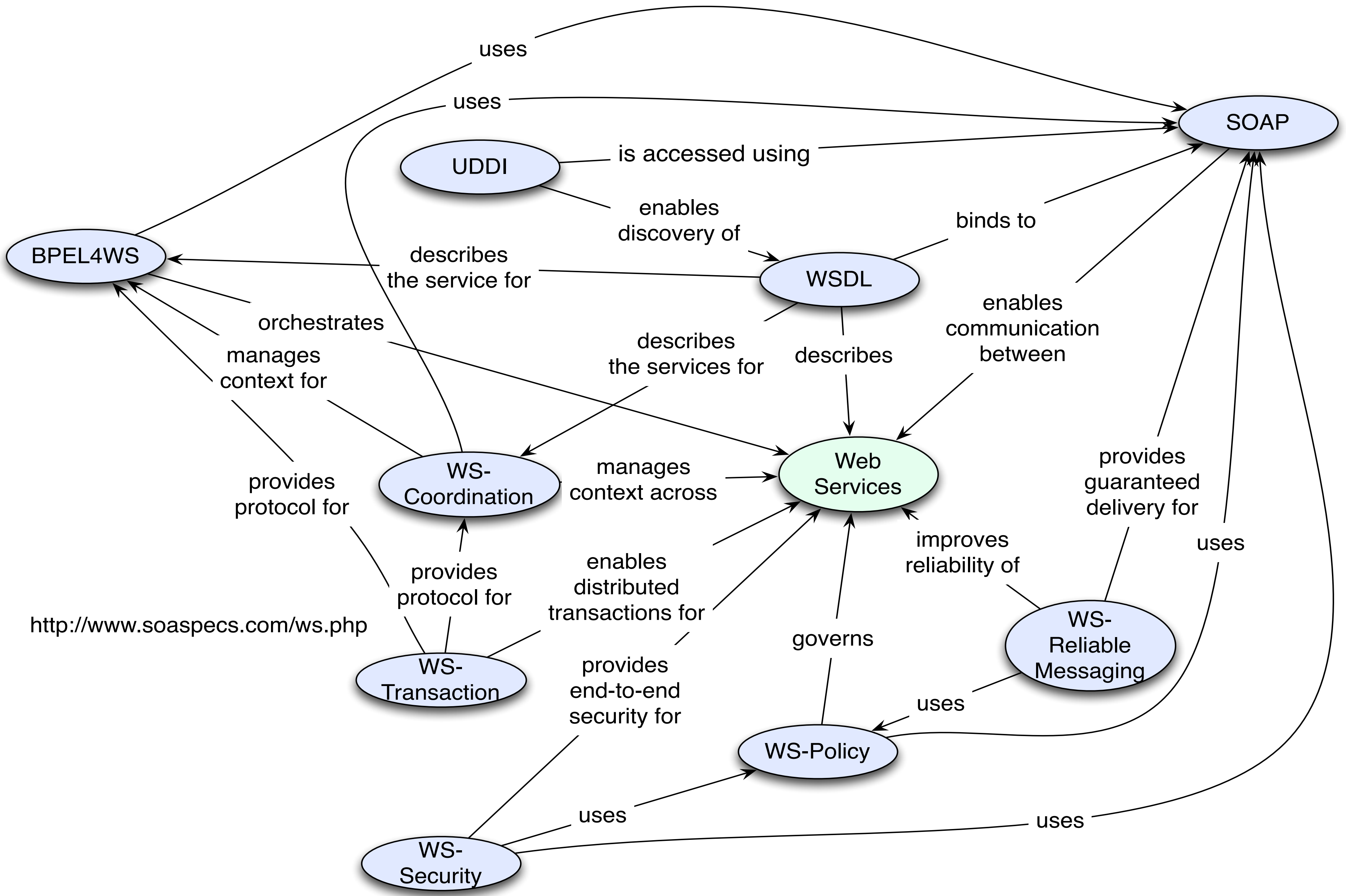
e.g., http://www.ibm.com/developerworks/webservices/standards/



Too many, Overlap, Conflicts, Convergence (WSDL, SOAP)

*(See ``**Interoperability Specifications**", May 06, IEEE Computer, by. Motahari, Benatallah, Casati and Toumani)*

WS-* extensions and their relationships



http://www.soaspecs.com/ws.php

WS-* extensions: Reliable Messaging (Blue Book Chap. 7)

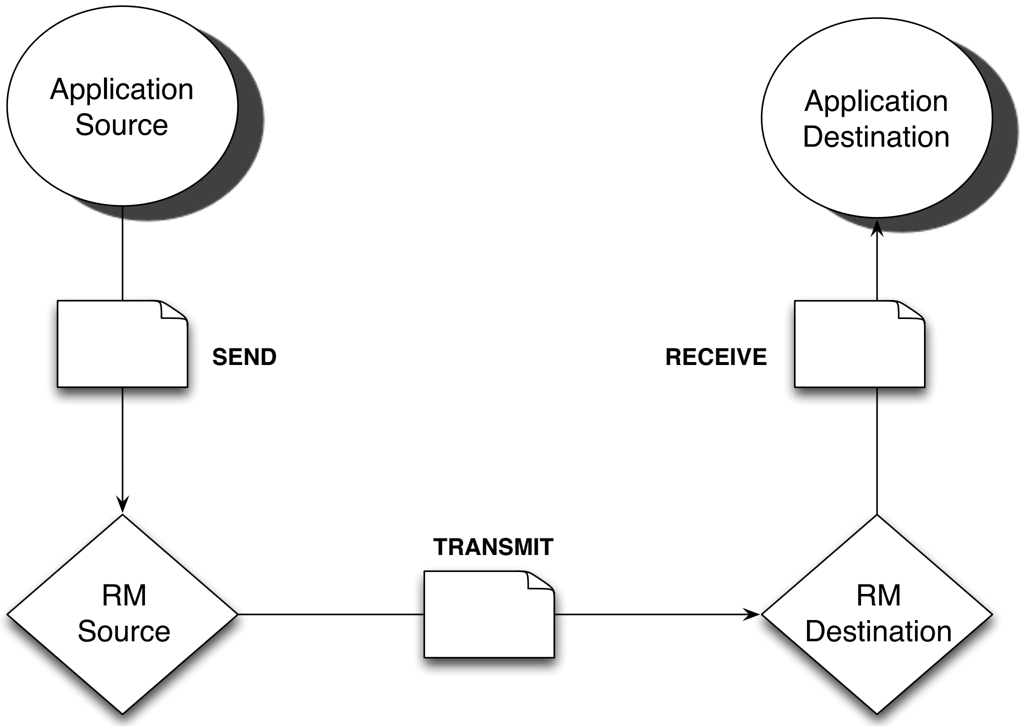
After a Web service transmit a message, it has no immediate way of knowing whether:

- the message successfully arrived at its destination
- the message failed to arrive and therefore requires a retransmission
- a series of messages arrived in the sequence they were intended to

Web service reliable messaging is a framework that enables an application running on one application server to reliably invoke a Web service running on another application server, assuming that both servers implement the WS-ReliableMessaging specification.

Reliable is defined as the ability to guarantee message delivery between the two endpoints (Web service and client) in the presence of software component, system, or network failures.

Reliable Messaging (Blue Book Chap. 7)



- WS-RM separates/abstracts ‘initiating messaging’ from ‘performing actual transmission’
- e.g., application source is the service that *sends* the message to the RM source (the physical processor/node that performs the actual wire *transmission*.

Reliable Messaging (Blue Book Chap. 7)

Sequences:

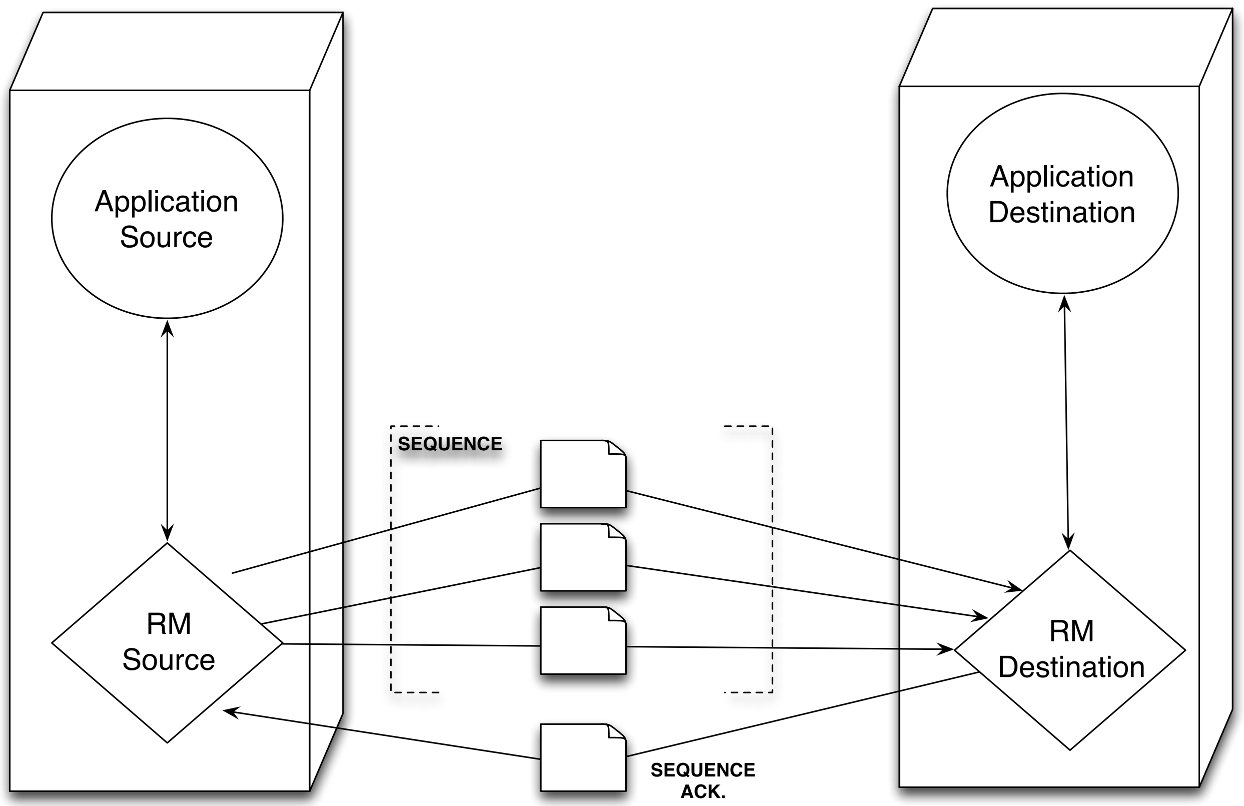
- A *sequence* establishes the order in which messages should be delivered
- Each message is labeled with a *message number*, the last one being a *last message* identifier.

Acknowledgements:

- A core part of reliable messaging is a notification system used to communicate conditions from the RM dest. to the RM source.
- The acknowledgement message indicates to the RM source which messages were received.

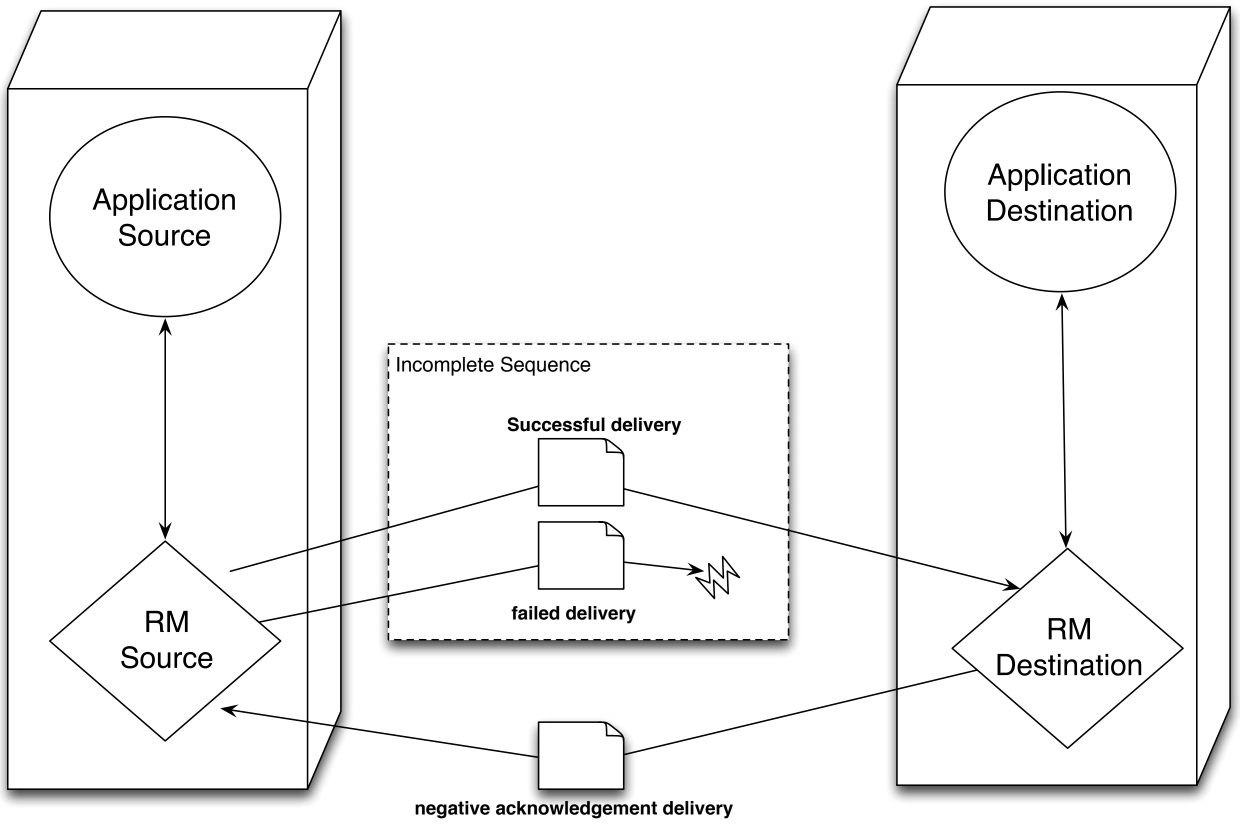
All this information is “injected” into SOAP headers within the messages themselves.

Reliable Messaging (Blue Book Chap. 7)



A sequence acknowledgement sent by the RM dest. after the successful delivery of a sequence of messages.

Reliable Messaging (Blue Book Chap. 7)

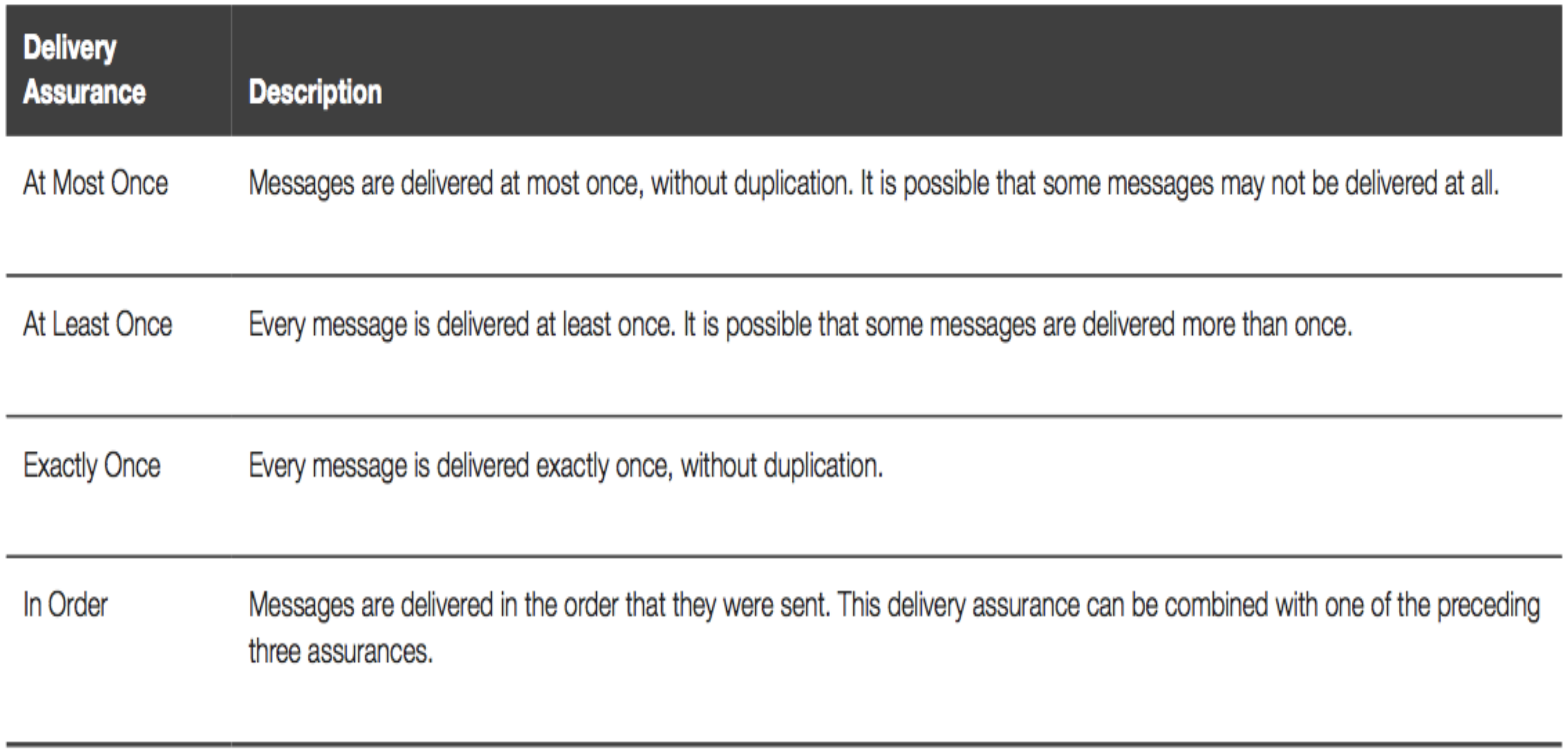


A negative ack. sent by the RM dest. to the RM source, indicating failed delivery prior to the completion of the sequence.

Reliable Messaging (Blue Book Chap. 7)

Delivery assurances:

- the nature of a sequence is determined by a set of reliability rules known as *Delivery Assurances*.
- They are predefined message delivery patterns that establish a set of reliability policies



Delivery Assurance	Description
At Most Once	Messages are delivered at most once, without duplication. It is possible that some messages may not be delivered at all.
At Least Once	Every message is delivered at least once. It is possible that some messages are delivered more than once.
Exactly Once	Every message is delivered exactly once, without duplication.
In Order	Messages are delivered in the order that they were sent. This delivery assurance can be combined with one of the preceding three assurances.

Reliable Messaging (Blue Book Chap. 17)

An example:

```
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsrm="http://schemas.xmlsoap.org/ws/2004/03/rm">
   <Header>
      <wsrm:Sequence>
         <wsu:Identifier>
            http://www.xmlrc.com/railco/seq22231
         </wsu:Identifier>
         <wsrm:MessageNumber>
            15
         </wsrm:MessageNumber>
         <wsrm:LastMessage/>
       </wsrm:Sequence>
   </Header>
   <Body>
    ...
   </Body>
</Envelope>
```

Reliable Messaging (Blue Book Chap. 17)

An example:

```
<Envelope
   xmlns="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
   xmlns:wsrm="http://schemas.xmlsoap.org/ws/2004/03/rm">
   <Header>
      <wsrm:SequenceAcknowledgement>
         <wsu:Identifier>
            http://www.xmlrc.com/railco/seq22231
         </wsu:Identifier>
         <wsrm:AcknowledgementRange Upper="4" Lower="1"/>
         <wsrm:AcknowledgementRange Upper="8" Lower="6"/>
         <wsrm:AcknowledgementRange Upper="12" Lower="11"/>
         <wsrm:AcknowledgementRange Upper="15" Lower="14"/>
      </wsrm:SequenceAcknowledgement>
   </Header>
   <Body>
    ...
   </Body>
</Envelope>
```