Slides for Chapter 9
Web Services

*From* **Coulouris, Dollimore, Kindberg and Blair**

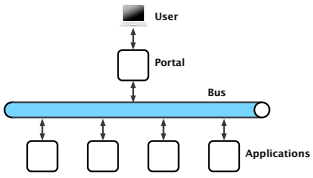
**Distributed Systems:**

**Concepts and Design**

Edition 5, © Addison-Wesley 2012

## Web services

- What are web services? A technology for application integration based on open standards (HTTP, XML)




- In what sense are they related to the Web (capital "W" to refer to the WWW)?
- In fact, deploying web services over the web is more of an artifact than a necessity ...
- What we care about is the web **of** services (services assembled from other services)



## Defining Web Service

- **Loosely coupled, document-based**
- Application functionality **packaged** as a single unit and **exposed** to the network
- First generation of web services: "simple", in the sense of non-composite, and closed (over existing, trusted relationships)
- Second Generation Services: Complex, and aggregated from web services provided by third parties (hence, open)

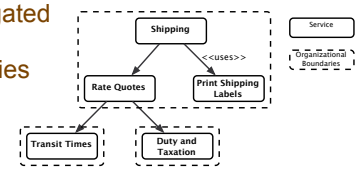
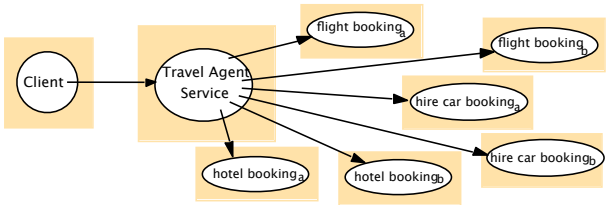
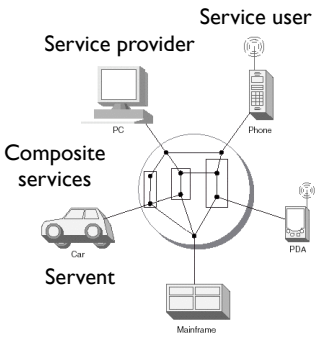


Figure 9.2
The 'travel agent service' combines other web services




## Web **of** Services

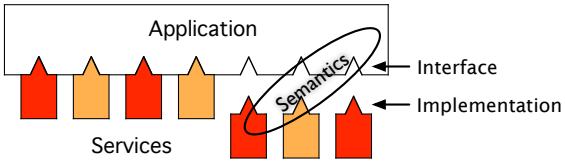
- In second generation services, applications are assembled from services **dynamically**
- Roles of service user and provider blend into what others call **servants** (eg in P2P)






## Vision

- Build your applications **on demand**
- Dynamically **discover** and **orchestrate** the execution of services on the network
- Will be able to choose between alternative implementations of the same service



## Web services vs. distributed objects

- Client-server interactions are very similar to RMI, but there are several subtle differences
- Remote object references vs. URIs
  - in distr. objects, objects can be created dynamically and uniquely. A service is a single object which can be accessed by several others
  - in web services, there is no object reference. Objects cannot be created dynamically. A web service is a single remote object, outside the control of the client
- RMI is usually internal to a single language (i.e., Java RMI is Java-to-Java). WS are open



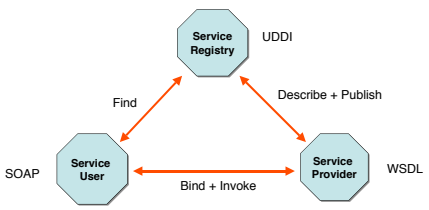
## Web services vs. distributed objects (cont.)

- In D.O., a service is a collection of objects (potentially remote and dynamically created), the servants. In WS, there are no servants, but just a server (like a single, persistent object)
- Distributed objects are on a par with local objects; web services require explicit handling
- Servants do not exist in web services
- Web services are intended to be used across organization boundaries; distributed objects are more intended for a single application within the same organization (cf. firewall issues).



## Service-Oriented Architecture

- Objectives:
  - Implementation transparency (common structure, neutral service description)
  - Location transparency (no hard binding, web-service agnostic interfaces)
- Roles:
  - Service Provider
  - Service User
  - Service Registry



SOA key aspects

- Combination of web services
  - programming at the level of web services
- Communication patterns
  - often synchronous request-replay, or event-based
  - more generally, several different paradigms can be used (differently from distributed objects)
- Loose coupling: minimization of dependencies between services
  - programming with interfaces
  - simple, minimal APIs (cf. REST)
  - coupling depends on the communication pattern

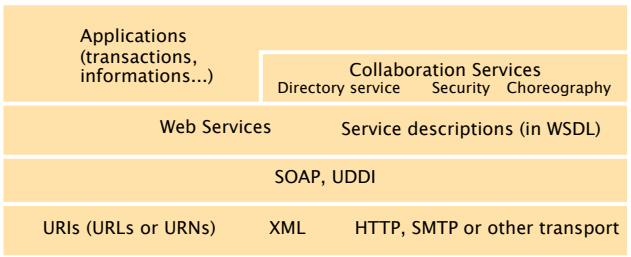


SOA key aspects

- Representation of messages
  - formal standard structure, extensible (typically XML)
- Service references
  - Based on web standards (URI, often simple URL)
- Activation of service
  - continuously or on demand - URL is persistent
- Transparency
  - SOA middleware do not hide data representation and marshalling to the programmer, but usually convenient APIs are provided, so that remote calls appear as local



Figure 9.1
Web services infrastructure and components




Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

## SOAP

- Simple Object Access Protocol
- Cross-platform **remote calls** (de facto, while technically also document exchange)
- Remote calls (typically) using HTTP as the transport and XML as the encoding
- Designed to be as simple as possible, so to make it easily understood and adopted
- (But REST is even simpler)
- But also complete enough to allow complex data structures to be transmitted



## REST [Fielding, 2000]

- Representional State Transfer
- A very constrained version of web services
- emphasis on manipulating remote resources, rather than on interfaces
- Each resource has its own URL, and is represented in XML
- clients use URLs and not URIs and only the HTTP operations GET, PUT, DELETE, POST to manipulate remote resources on servers
- (On Amazon Web Services, more than 80% of web service requests are via REST)



## SOAP Messages

- SOAP messages are XML documents usually sent over HTTP with a certain format


- Components of a SOAP message:

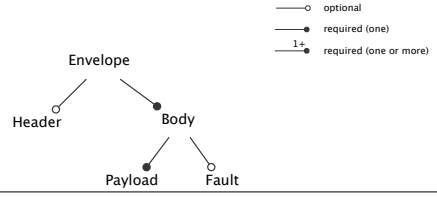
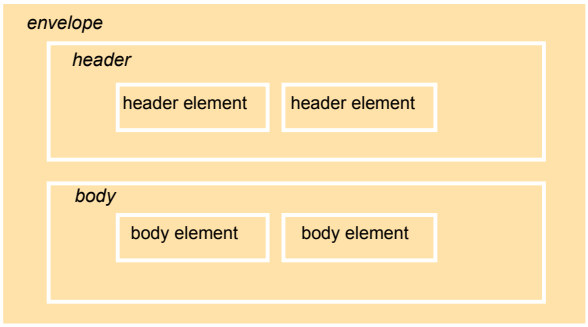
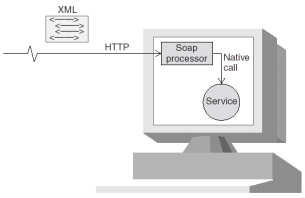


Figure 9.3
SOAP message in an envelope




## SOAP containers

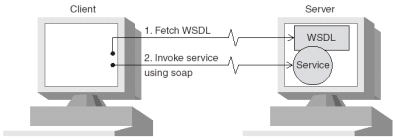
- Container accepts incoming requests and dispatches them to the service
- Translates between SOAP and the native language of the service (Java, C#, ...)
- Clients only need to know the address of the service, and messages it understands, but not what language, platform, or location



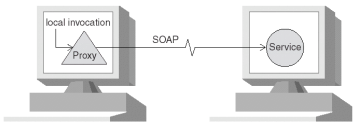



## Binding to a Service

- Clients get address and messages from a WSDL description of the service




- The binding process (dynamically) returns a proxy to the remote web service

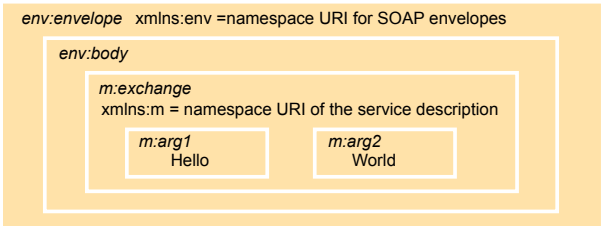


WSDL

- Web Service Description Language
- Neutral format for services to advertise themselves on the network
- In future, we can choose between competing providers of same service (price, ...)
- Components of a WSDL description:
  - Service
  - Ports
  - Operations
  - Messages
- Generating a WSDL description



Figure 9.4
Example of a simple request without headers




In this figure and the next, each XML element is represented by a shaded box with its name in italic followed by any attributes and its content

Figure 9.5
Example of a reply corresponding to the request in Figure 9.4

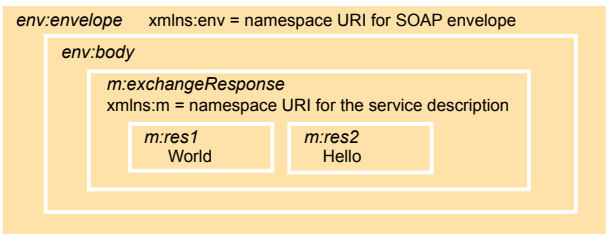
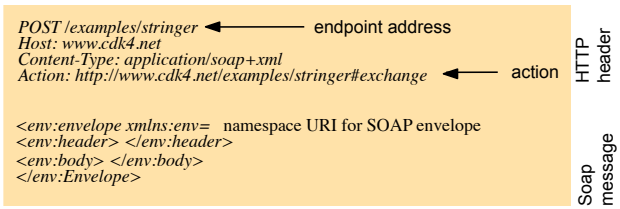


Figure 9.6
Use of HTTP POST Request in SOAP client-server communication





© Pearson Education 2012

## Reliable Communication

- SOAP uses TCP (usually)
- OASIS has defined WS-ReliableMessaging, with these semantics
  - At least once: message is delivered at least once, but an error is reported if it cannot be delivered
  - At most once: message is delivered at most once, with no errors if it cannot be delivered
  - Exactly once: message is delivered exactly once, and an error is reported if it cannot be delivered
- Ordering of messages can be provided in combination with any of the above



Figure 9.7
Java web service interface ShapeList

```
import java.rmi.*;
public interface ShapeList extends Remote {
        int newShape(GraphicalObject g) throws RemoteException; 1
        int numberOfShapes()throws RemoteException;
        int getVersion() throws RemoteException;
        int getGOVersion(int i)throws RemoteException;
        GraphicalObject getAllState(int i) throws RemoteException;
}
```


© Pearson Education 2012

Figure 9.8
Java implementation of the ShapeList server

```
import java.util.Vector;
public class ShapeListImpl implements ShapeList {
        private Vector theList = new Vector();
        private int version = 0;
        private Vector theVersions = new Vector();
        public int newShape(GraphicalObject g) throws RemoteException{
                version++;
                theList.addElement(g);
                theVersions.addElement(new Integer(version));
                return theList.size();
        }
        public int numberOfShapes(){}
        public int getVersion() {}
        public int getGOVersion(int i){  }
        public GraphicalObject getAllState(int i) {}
}
```

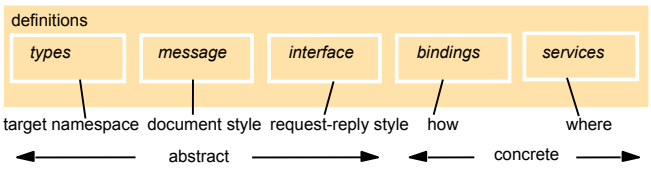

© Pearson Education 2012

Figure 9.9
Java implementation of the *ShapeList* client

```
package staticstub;
import javax.xml.rpc.Stub;
public class ShapeListClient {
        public static void main(String[] args) { /* pass URL of service */
           try {
                Stub proxy = createProxy();                                    1
                proxy._setProperty                                             2
                   (javax.xml.rpc.Stub.ENDPOINT_ADDRESS_PROPERTY, args[0]);
                ShapeList aShapeList = (ShapeList)proxy;                       3
                GraphicalObject g = aShapeList.getAllState(0);                 4
           } catch (Exception ex) { ex.printStackTrace(); }
        }
        private static Stub createProxy() {                                    5
                return
                   (Stub) (new MyShapeListService_Impl().getShapeListPort());  6
   }
}
```


© Pearson Education 2012

Figure 9.10
The main elements in a WSDL description





© Pearson Education 2012

Figure 9.11
WSDL request and reply messages for the newShape operation

```
message name =  "ShapeList_newShape"
    part  name ="GraphicalObject_1"
         type =  "ns:GraphicalObject "

message name = "ShapeList_newShapeResponse"
    part  name=  "result"
         type=  "xsd:int"
```

tns – target namespace  xsd – XML schema definitions


© Pearson Education 2012

Figure 9.12
Message exchange patterns for WSDL operations

| *Name* | *Messages sent by* | | | |
|---|---|---|---|---|
| | *Client* | *Server* | *Delivery* | *Fault message* |
| In-Out | *Request* | *Reply* | | may replace *Reply* |
| In-Only | *Request* | | | no fault message |
| Robust In-Only | *Request* | | guaranteed | may be sent |
| Out-In | *Reply* | *Request* | | may replace *Reply* |
| Out-Only | | *Request* | | no fault message |
| Robust Out-Only | | *Request* | guaranteed | may send fault |


© Pearson Education 2012

Figure 9.13
WSDL operation newShape

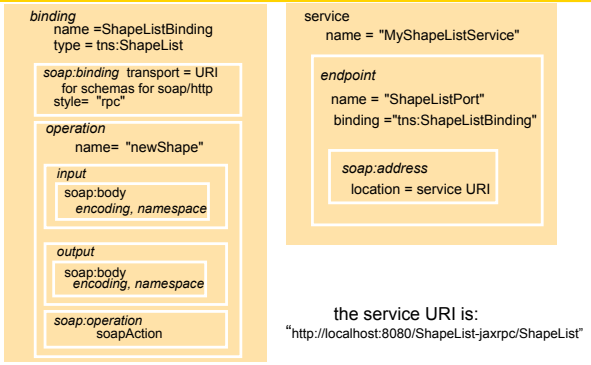
```
operation name =  "newShape"
    pattern = In-Out
  input message = tns:ShapeList_newShape
  output message = "tns:ShapeList_newShapeResponse"
```

tns – target namespace xsd – XML schema definitions

The names *operation*, *pattern*, *input* and *output* are defined in the XML schema for WSDL


© Pearson Education 2012

Figure 9.14
SOAP binding and service definitions




Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

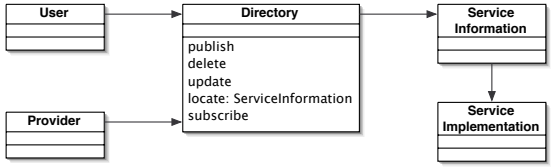
## UDDI

- Universal Discovery and Directory Interface
- Service discovery protocol
- Kind of *yellow pages* which allows applications to obtain information about businesses and their web services
- Helps potential business partners to find each others' business services
- A system will have *some* kind of directory, but not always need all UDDI features



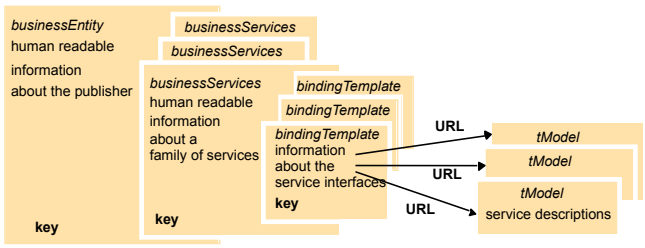
## Service Directory

- A service directory allows publishers to publish information about their services, and users to locate them (and receive updates)




- UDDI stores service information in the form of business entities, business services, binding templates and technical models

Figure 9.15
The main UDDI data structures




Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

## XML Security

- XML security is a set of W3C designs for signing, key management and encryption
- Intended for cooperative work in documents whose contents may need to be authenticated and/or encrypted (also partially)
- Can be applied to SOAP, since messages are in XML
  - Another approach, SOAP specific: WS-Security
- Example: a patient's medical records ("fascicolo sanitario")
- These needs cannot be dealt with at lower levels (e.g. SSL/TLS)



## XML security requirements

- Encrypt a whole document, or some selected parts
  - some parts may be sensible, others may not
- Sign a whole document, of some selected parts
  - some parts may not need, or cannot, be signed
- Add to a signed document, and sign the result
- Authorize different users to see different parts
  - e.g. the patient's folder
- In fact, any part of the XML document can be signed and encrypted on its own

## XML-Signature Syntax and Processing (W3C, 2002)

•The following example is a detached signature of the content of the HTML4 in XML specification.

```
1.  <Signature Id="MyFirstSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
2.    <SignedInfo>
3.      <CanonicalizationMethod Algorithm="http://www.w3.org/TR/2001/REC-xml-
      c14n-20010315"/>
4.      <SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#dsa-sha1"/>
5.      <Reference URI="http://www.w3.org/TR/2000/REC-xhtml1-20000126/">
6.        <Transforms>
7.          <Transform Algorithm="http://www.w3.org/TR/2001/REC-xml-c14n-20010315"/>
8.        </Transforms>
9.        <DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
10.       <DigestValue>j6lwx3rvEPO0vKtMup4NbeVu8nk=</DigestValue>
11.     </Reference>
12.   </SignedInfo>
13.   <SignatureValue>MC0CFFrVLtRlk=...</SignatureValue>
14.     <KeyInfo>
15.       <KeyValue>
16.         <DSAKeyValue>
17.           <P>...</P><Q>...</Q><G>...</G><Y>...</Y>
18.         </DSAKeyValue>
19.       </KeyValue>
20.     </KeyInfo>
21.  </Signature>
```



## Figure 9.16
## Algorithms required for XML signature

| *Type of algorithm* | *Name of algorithm* | *Required* | *reference* |
|---|---|---|---|
| Message digest | SHA-1 | Required | Section 7.4.3 |
| Encoding | base64 | Required | [Freed and Borenstein 1996] |
| Signature | DSA with SHA-1 | Required | [NIST 1994] |
| (asymmetric) | RSA with SHA-1 | Recommended | Section 7.3.2 |
| MAC signature (symmetric) | HMAC-SHA-1 | Required | Section 7.4.2 and Krawczyk *et al.* [1997] |
| Canonicalization | Canonical XML | Required | Page 810 |

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

## XML Encryption Syntax and Processing (W3C, 2002)

- One may require to encrypt
  - a single XML element
  - an element content (elements or character data)
  - arbitrary XML documents as a whole
  - other encrypted data ("super encryption")

- Consider the following fictitious payment information:

```
1.  <?xml version='1.0'?>
2.  <PaymentInfo xmlns='http://example.org/paymentv2'>
3.    <Name>John Smith</Name>
4.    <CreditCard Limit='5,000' Currency='USD'>
5.      <Number>4019 2445 0277 5567</Number>
6.      <Issuer>Example Bank</Issuer>
7.      <Expiration>04/02</Expiration>
8.    </CreditCard>
9.  </PaymentInfo>
```

- Several sensitive informations, different possible "views"

## XML Encryption Syntax and Processing (W3C, 2002)

• Encrypting an XML Element: Smith's credit card number is sensitive information!

```
1.  <?xml version='1.0'?>
2.  <PaymentInfo xmlns='http://example.org/paymentv2'>
3.    <Name>John Smith</Name>
4.    <EncryptedData Type='http://www.w3.org/2001/04/xmlenc#Element'
5.     xmlns='http://www.w3.org/2001/04/xmlenc#'>
6.      <CipherData>
7.        <CipherValue>A23B45C56</CipherValue>
8.      </CipherData>
9.    </EncryptedData>
10. </PaymentInfo>
```

• Encrypting Arbitrary Data and XML Documents

```
1.  <?xml version='1.0'?>
2.  <EncryptedData xmlns='http://www.w3.org/2001/04/xmlenc#'
3.   MimeType='text/xml'>
4.    <CipherData>
5.      <CipherValue>A23B45C56</CipherValue>
6.    </CipherData>
7.  </EncryptedData>
```



## Figure 9.17
## Algorithms required for encryption(in Figure 9.16 are also required)

| *Type of algorithm* | *Name of algorithm* | *Required* | *reference* |
|---|---|---|---|
| Block cipher | TRIPLEDES,<br>AES 128<br>AES-256<br>AES-192 | required<br><br><br>optional | Section 7.3.1 |
| Encoding | base64 | required | [Freed and Borenstein 1996] |
| Key transport | RSA-v1.5,<br>RSA-OAEP | required | Section 7.3.2<br>[Kaliski and Staddon 1998] |
| Symmetric key wrap (signature by shared key) | TRIPLEDES KeyWrap,<br>AES-128 KeyWrap,<br>AES 256KeyWrap<br>AES-192 KeyWrap | required<br><br><br>optional | [Housley 2002] |
| Key agreement | Diffie-Hellman | optional | [Rescorla, 1999] |


© Pearson Education 2012

## Figure 9.19
## A selection of Amazon Web Services

| *Web service* | *Description* |
|---|---|
| Amazon Elastic Compute Cloud (EC2) | Web-based service offering access to virtual machines of a given performance and storage capacity |
| Amazon Simple Storage Service (S3) | Web-based storage service for unstructured data |
| Amazon Simple DB | Web-based storage service for querying structured data |
| Amazon Simple Queue Service (SQS) | Hosted service supporting message queuing (as discussed in Chapter 6) |
| Amazon Elastic MapReduce | Web-based service for distributed computation using the MapReduce model (introduced in Chapter 21) |
| Amazon Flexible Payments Service (FPS) | Web-based service supporting electronic payments |


© Pearson Education 2012

## Coordination of web services

- SOAP supports single request-response interactions
- Real applications implement services by means of many requests to several other services
  - Order may be important
  - Consistent states in servers
  - Possibility of aborting requests
- The client (the web service) must follow some protocol in interacting with other web services
- Distributed transactions
  - Two-phase commit protocols (Chap. 17)
  - Long time sessions: contracts with compensations



## Figure 9.18
## Travel agent scenario

1. The client asks the travel agent service for information about a set of services; for example, flights, car hire and hotel bookings.
2. The travel agent service collects prices and availability information and sends it to the client, which chooses one of the following on behalf of the user:

(a) refine the query, possibly involving more providers to get more information, then repeat step 2;

(b) make reservations;

(c) quit.

3. The client requests a reservation and the travel agent service checks availability.
4. Either all are available;
   or for services that are not available;
   either alternatives are offered to the client who goes back to step 3;
   or the client goes back to step 1.
5. Take deposit.
6. Give the client a reservation number as a confirmation.
7. During the period until the final payment, the client may modify or cancel reservations

Instructor's Guide for Coulouris, Dollimore, Kindberg and Blair, Distributed Systems: Concepts and Design Edn. 5
© Pearson Education 2012

## Web service Choreography

- Lot of work on a general model for web service coordination
  - eg, for defining distributed transactions, ensuring deadlock freedom, etc.
- In principle, it could be possible to describe the valid alternative interactions between each pair of web services
- This can be used for defining the coordination of several web services in the joint task
- **Orchestration**: automated arrangement, coordination, and management of services
- a local view from the perspective of one participant, called the **orchestrator**

## Web service Choreography

- **Service choreography**: service composition in which the interaction protocol is defined from a global perspective
- A choreography shows the behavior of each member of a set of participants, but does not need a central controller
- Intuition: “Dancers dance following a global scenario without a single point of control"
- Several languages proposed for choreographies (e.g. BPEL4WS, BPMN, WSFL, WS-CDL…)
- New languages like **Jolie** use typing systems for representing valid interactions (and hence global types can be seen as choreographies)