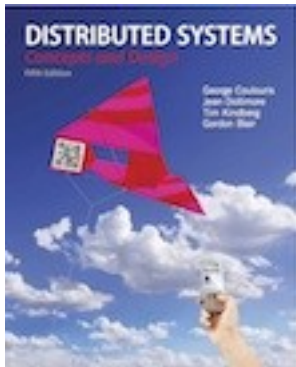


Slides for Chapter 9

Web Services



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

Distributed Systems: Concepts and Design

Edition 5, © Addison-Wesley 2012

Figure 9.1 Web services infrastructure and components

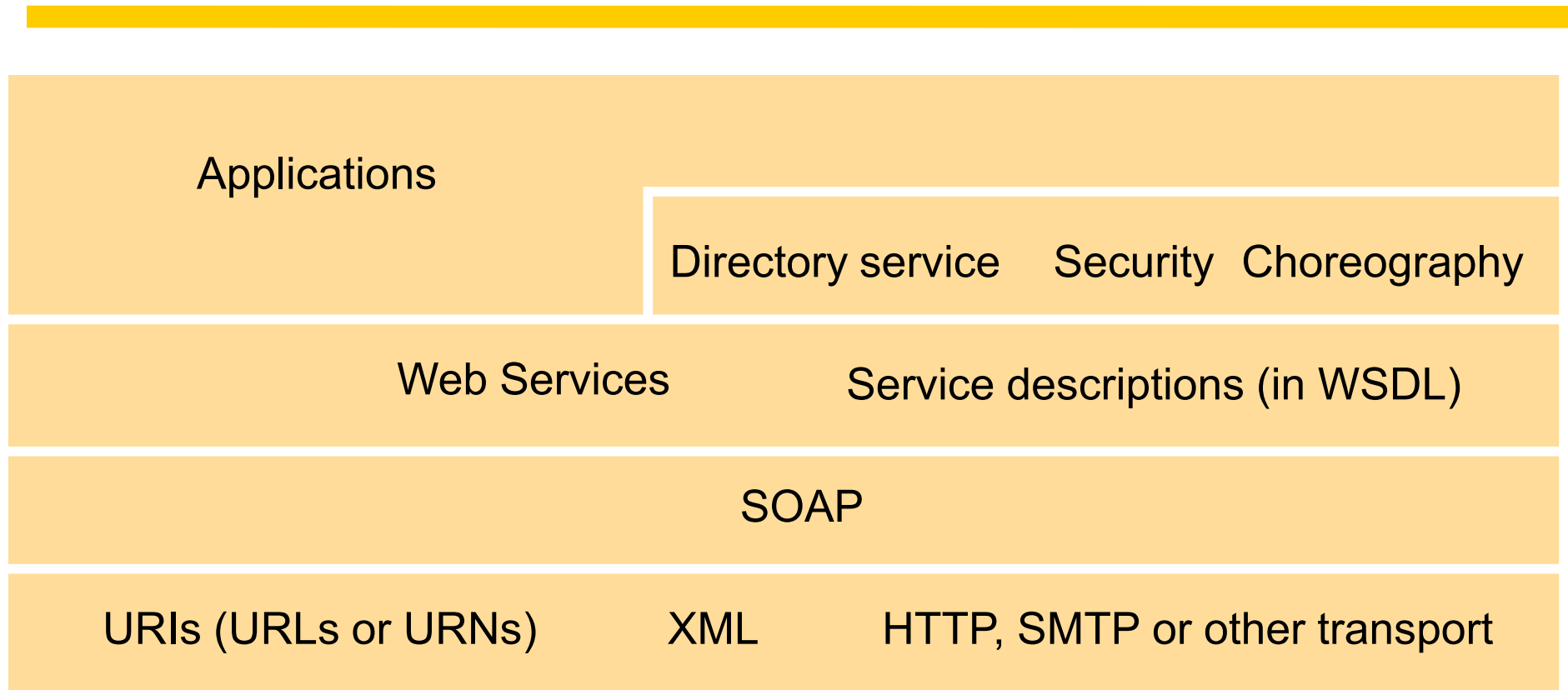


Figure 9.2

The 'travel agent service' combines other web services

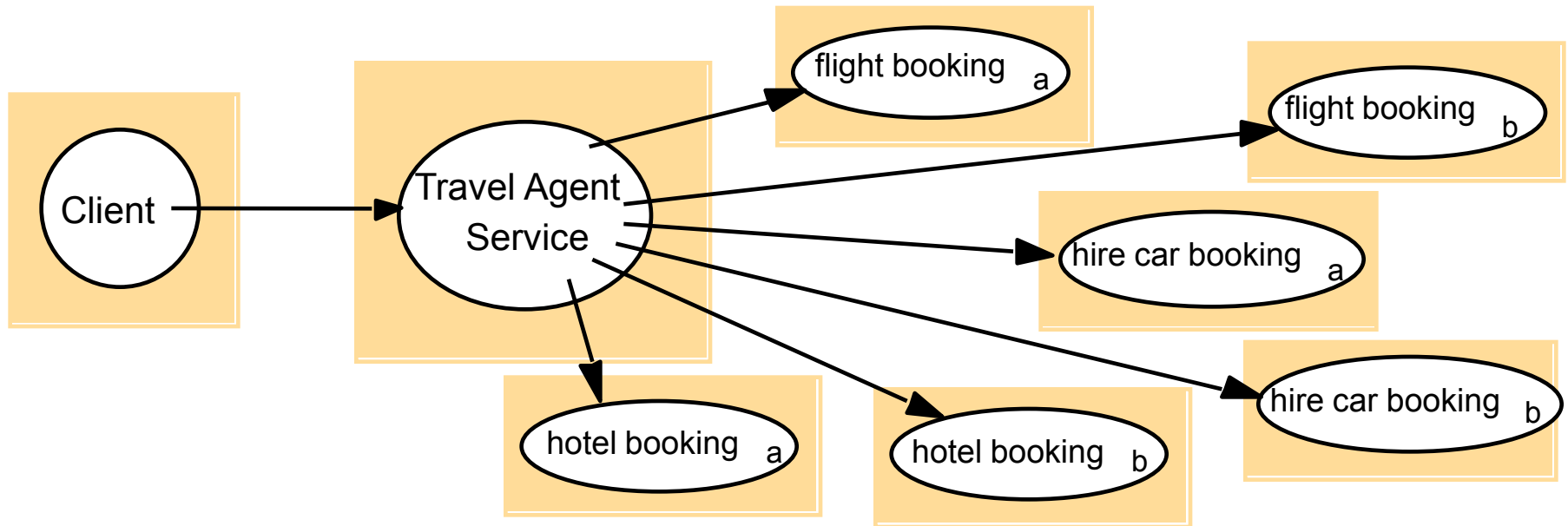


Figure 9.3 SOAP message in an envelope

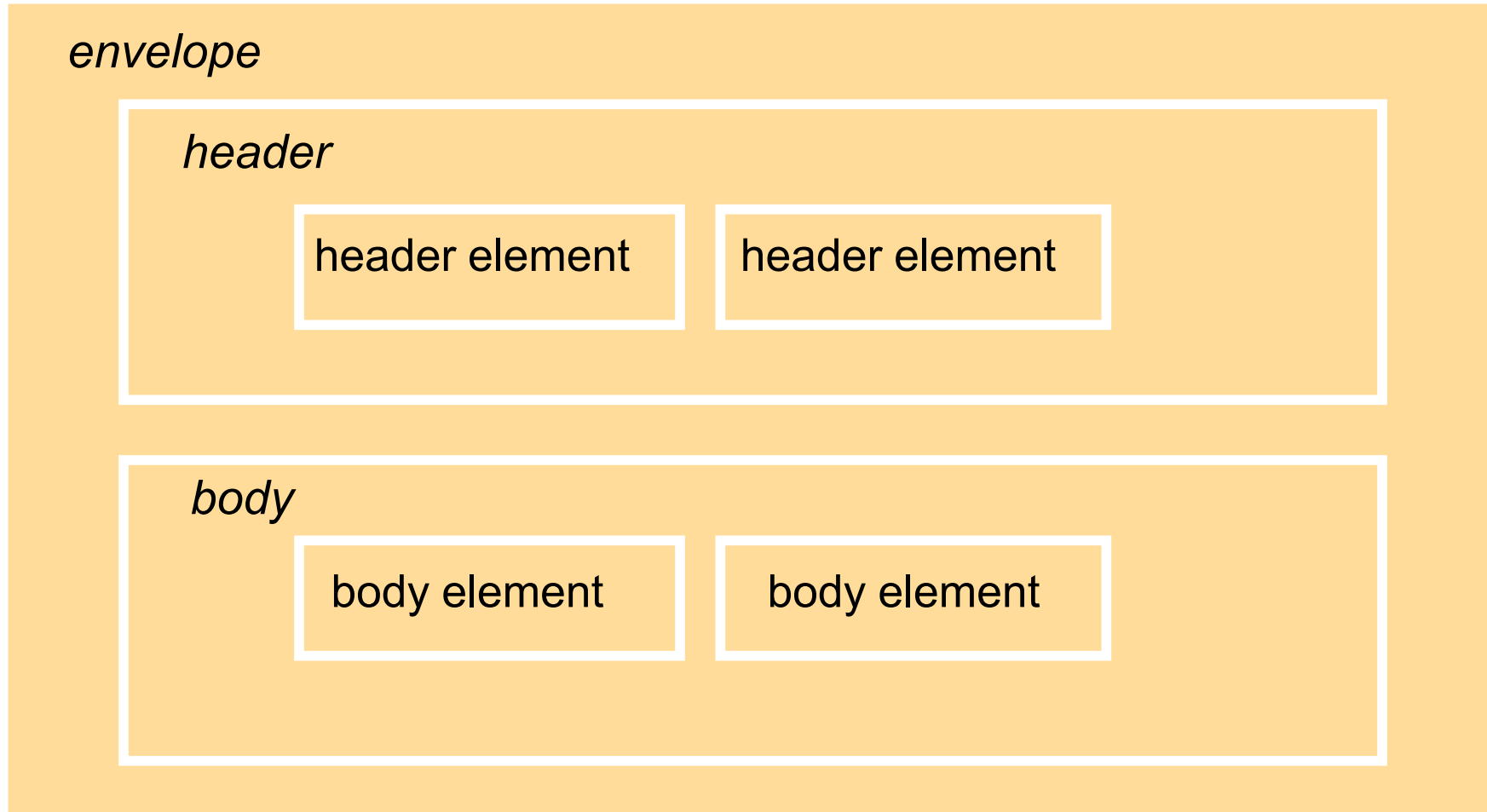


Figure 9.4

Example of a simple request without headers

env:envelope xmlns:env = namespace URI for SOAP envelopes

env:body

m:exchange

xmlns:m = namespace URI of the service description

m:arg1
Hello

m:arg2
World

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

env:envelope xmlns:env = namespace URI for SOAP envelope

env:body

m:exchangeResponse

xmlns:m = namespace URI for the service description

m:res1
World

m:res2
Hello

Figure 9.6

Use of HTTP POST Request in SOAP client-server communication

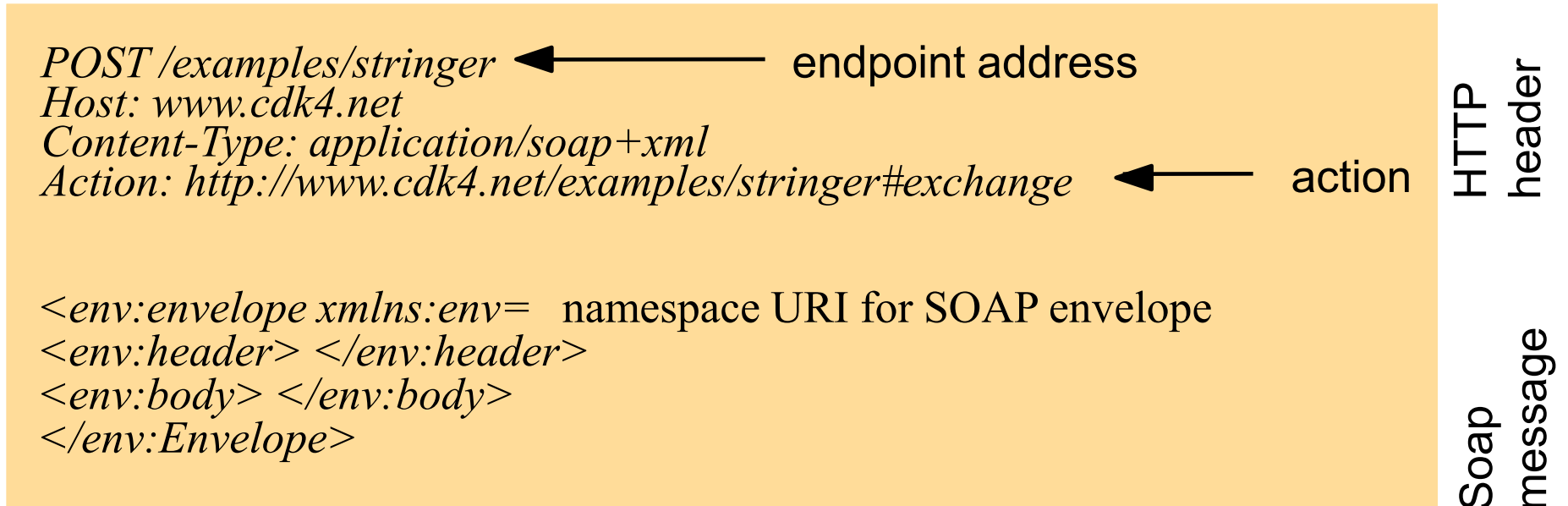


Figure 9.7

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;  
}
```


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) {}  
}
```

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
    }
}
```

Figure 9.10
The main elements in a WSDL description

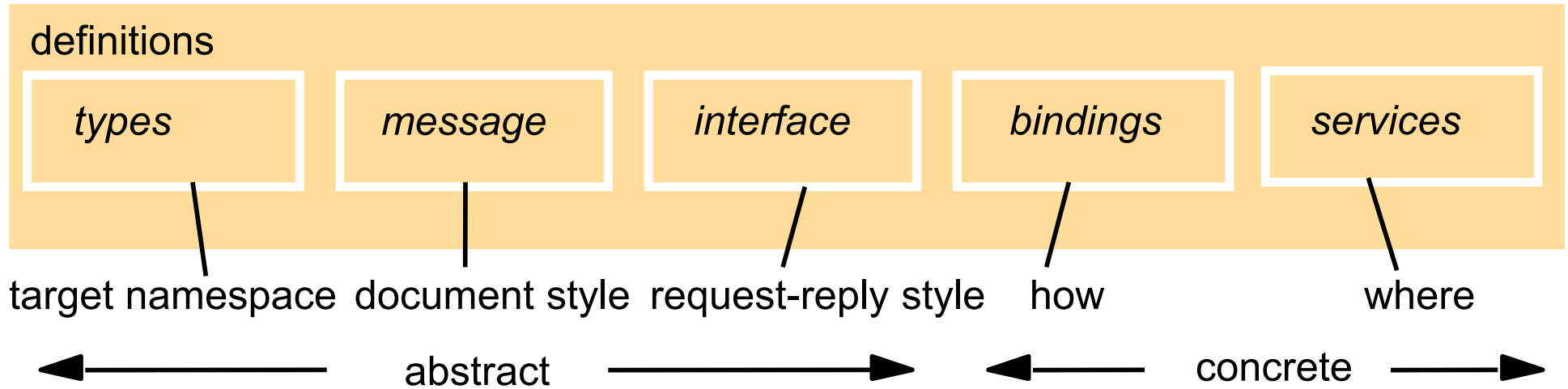


Figure 9.11

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

Figure 9.12

Message exchange patterns for WSDL operations

<i>Name</i>	<i>Messages sent by</i>			
	<i>Client</i>	<i>Server</i>	<i>Delivery</i>	<i>Fault message</i>
In-Out	<i>Request</i>	<i>Reply</i>		may replace <i>Reply</i>
In-Only	<i>Request</i>			no fault message
Robust In-Only	<i>Request</i>		guaranteed	may be sent
Out-In	<i>Reply</i>	<i>Request</i>		may replace <i>Reply</i>
Out-Only		<i>Request</i>		no fault message
Robust Out-Only		<i>Request</i>	guaranteed	may send fault

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

Figure 9.14 SOAP binding and service definitions

binding

name = ShapeListBinding
type = tns:ShapeList

soap:binding transport = URI
for schemas for soap/http
style= "rpc"

operation

name= "newShape"

input

soap:body
encoding, namespace

output

soap:body
encoding, namespace

soap:operation

soapAction

service

name = "MyShapeListService"

endpoint

name = "ShapeListPort"

binding = "tns:ShapeListBinding"

soap:address

location = service URI

the service URI is:

“http://localhost:8080/ShapeList-jaxrpc/ShapeList”

Figure 9.15
The main UDDI data structures

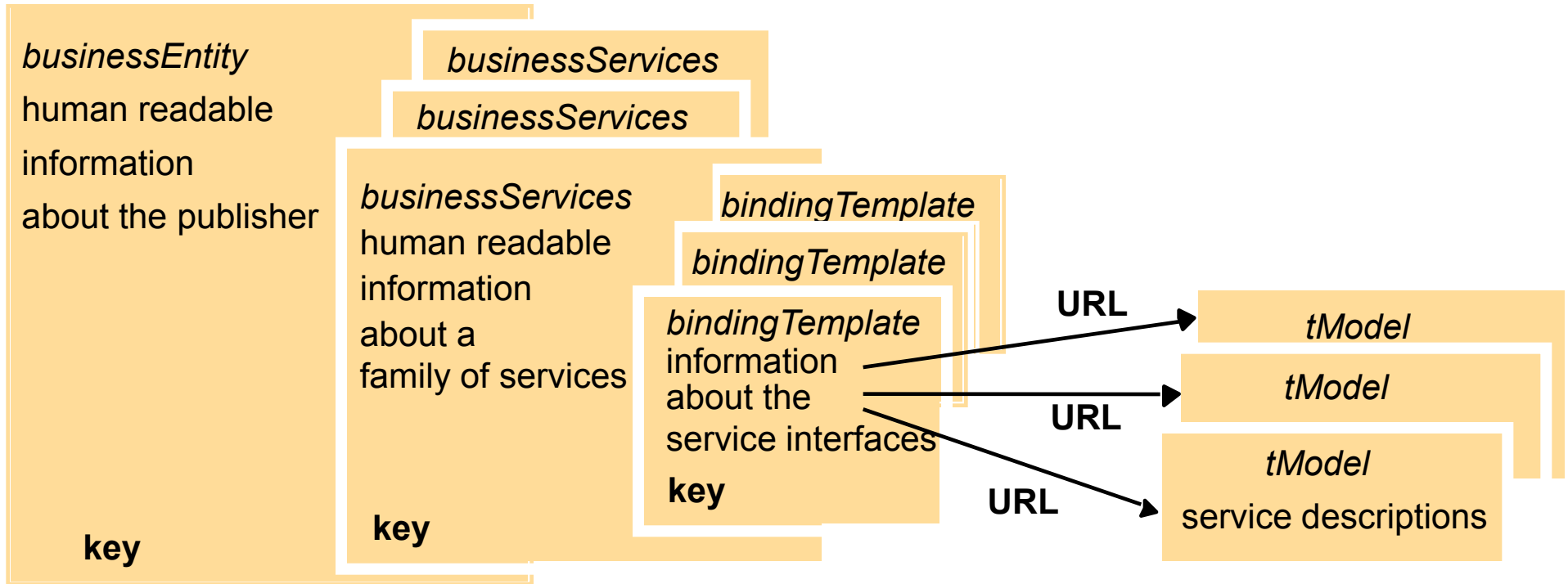


Figure 9.16

Algorithms required for XML signature

<i>Type of algorithm</i>	<i>Name of algorithm</i>	<i>Required</i>	<i>reference</i>
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 <i>et al.</i> [1997]
Canonicalization	Canonical XML	Required	Page 810

Figure 9.17

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

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

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

Figure 9.19

A selection of Amazon Web Services

<i>Web service</i>	<i>Description</i>
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