1DT066
DISTRIBUTED INFORMATION SYSTEM

Middleware

OUTLINE

- Middleware
- RPCs
- Web Services
- Summary

1. Middleware

- Middleware is a computer software that *connects software components* or people and their applications.
- It consists of a set of services that allows multiple processes running on one or more machines to interact.
- ***Interoperable*** in support of distributed systems.
- Middleware sits "in the middle" between application software that may be working on different operating systems.

1.Conceptual Framework

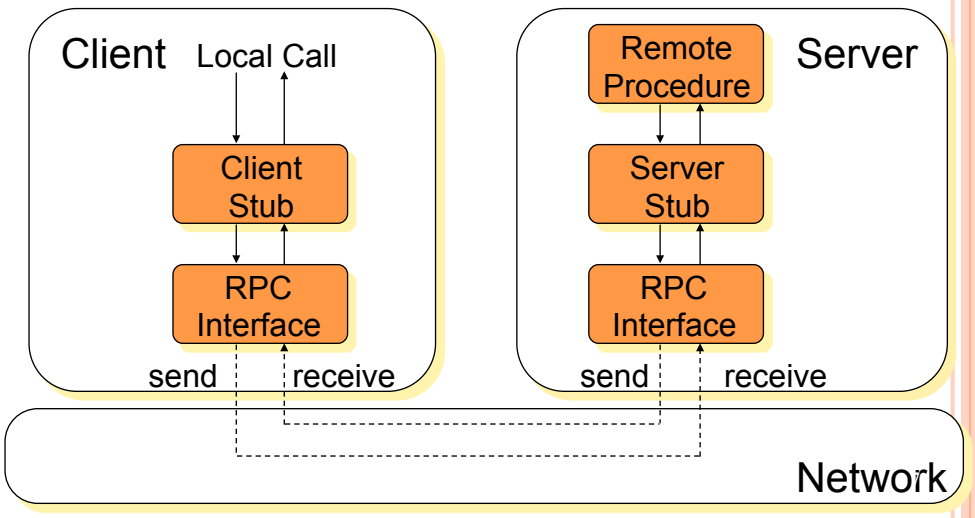
- Architecture.
- Accessing components from programming languages.
- Interfaces to lower layers.
- Component identification (to achieve *location transparency*).
- Service invocation styles.
- Handling of failures.

Part I: RPC

2 Remote Procedure Calls

- Overview of RPC architecture.
- Generation of client/server stubs.
- RPC interface.
- Binding.
- Handling of remote procedures.

2.1 RPC Architecture



2.2 The RPC Language

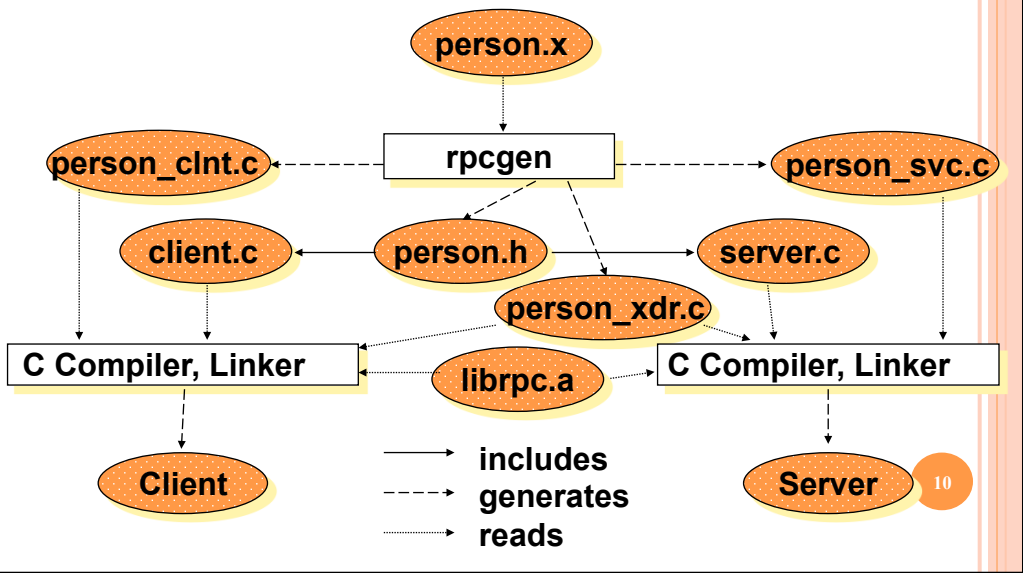
- Definition of types (similar to C).
- Component is described as a `PROGRAM`.
- `PROGRAM` has an identification and a version number.
- `PROGRAM` exports procedures
  - Procedures have a result type and a parameter list,
  - Procedure can be called from remote components,
  - Call can be defined statically or dynamically.

2.2 The RPC Language (Example)

```
/* person.x */
const NL=64;
enum sex_type {
 FEMALE = 1,MALE = 2
};
struct Person {
  string first_name<NL>;
  string last_name<NL>;
  sex_type sex;
  string city<NL>;
};
```

```
program PERSONPROG {
 version PERSONVERS {
  void PRINT(Person)=0;
  int STORE(Person)=1;
  Person LOAD(int)=2;
 } = 0;
} = 105040;
```


2.2 Generation of Stubs



2.2 Implementation of Server

```
/* server.c */
void * print_0(Person *argp,
                    struct svc_req * rqstp)
{
  static char * result;
  printf("%s %s\n%s\n\n",
         argp->first_name,
         argp->last_name,
         argp->city);
  return((void *) &result);
}
```


2.2 Use of Stubs

```
/* client.c */
print_person(char * host, Person * pers) {
   ...
   if (print_0(pers, clnt)==NULL)
     /* call failed /*
   ...
}
```

2.3 RPC Interface

- Used by client or server directly:
  - Locating servers.
  - Choosing a transport protocol.
  - Authentication and security.
  - Invoking RPCs dynamically.
- Used by stubs for:
  - Generating unique message IDs.
  - Sending messages.
  - Maintaining message history.

2.3 RPC Interface

```
print_person(char * host, Person * pers) {
  CLIENT *clnt;
  clnt = clnt_create(host, PERSONPROG,
                     PERSONVERS, "udp");
  if (clnt == (CLIENT *) NULL) {
     exit(1);
  }
  if (print_0(pers, clnt)==NULL)
    clnt_perror(clnt, "call failed");
  clnt_destroy(clnt);
}
```

2.4 Binding

- How to locate an RPC server that can execute a given procedure in a network?
- Can be done
  - statically (i.e. at compile-time) or
  - dynamically (i.e. at run-time).
- Dynamic binding is supported by portmap daemons.

2.5 Handling of Remote Procedures

- Call handled synchronously by server.
- Concurrent RPCs:
  - serial or
  - concurrently.
- Server availability:
  - continuous or
  - on-demand.

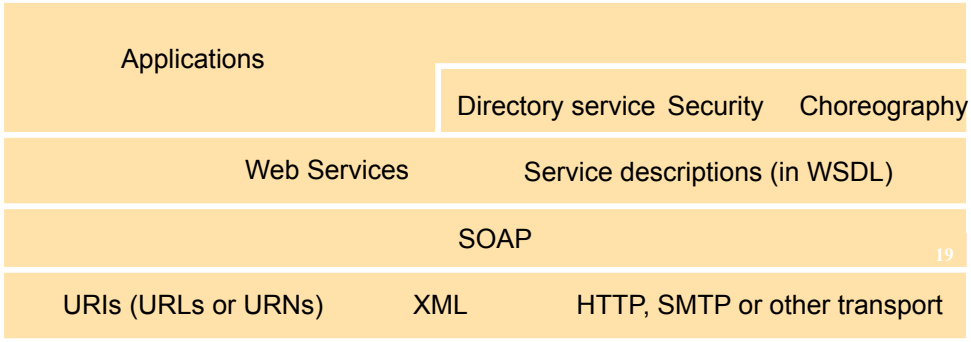
Part II: Web Services

3. Web Services

- Introduction
- Web Services
- Service descriptions and IDL for web services
- A directory service for use with web services
- Coordination of web services

3.1 INTRODUCTION

- Simple protocol restricts the potential scope of application.
- Web services provide an infrastructure for maintaining a richer and more structured form of interoperability between clients and servers.

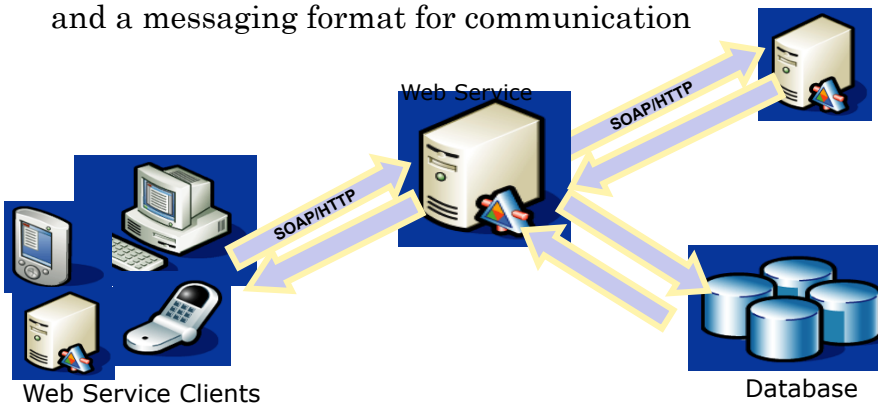


3.2.1 WEB SERVICES

- A web service interface generally consists of a collection of operations that can be used by clients over the Internet.
- The key characteristic of most web services is they can process XML-formatted SOAP message
- Properties:
  - It is accessible over the Web.
  - It provides an interface that can be called from another program
  - It is registered and can be located through a Web Service registry.
  - It communicates using message over standard Web protocols.

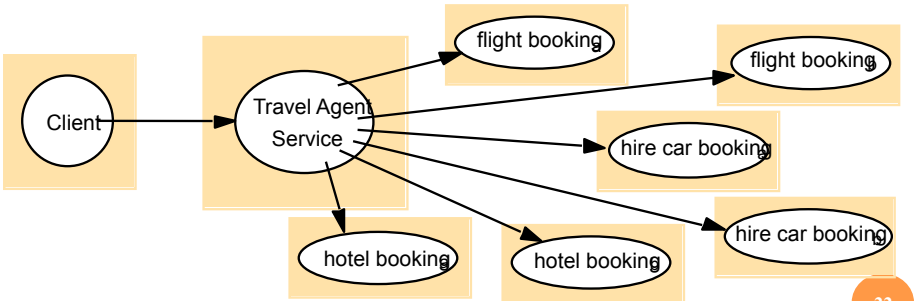
3.2.2 Web Services

- Loosely-coupled software applications that use open standards to describe an interface for accessing them and a messaging format for communication



3.2.3 Characteristic of Web Services

- Combination of web services
  - Allows its operations to be combined with those of other services to provide new functionality.



Example: The 'travel agent service' combines other web services

3.2.3 CHARACTERISTIC OF WEB SERVICES

- Two communication patterns
  - Processing of a booking takes a long time to complete and could well be supported by an *asynchronous* exchange of documents
  - The checking of credit card details and the interactions with the client should be supported by a *synchronous* request-reply protocol.
- No particular programming model
  - They are independent of any particular programming paradigm.

3.2.3 CHARACTERISTIC OF WEB SERVICES

- Representation of messages
  - Both SOAP message and the data it carries are represented in XML
- Services references
  - Each web service has a URL, which clients use to access the service.
- Transparency
  - The details of SOAP and XML are generally hidden by a local API in a programming language. The service description may be used as a basis for automatically generating the necessary marshalling and unmarshalling procedures.

3.2.4 Web Services Core Technologies

- Technologies for Web Services
  - XML
  - SOAP (XML Based)
  - WSDL (XML Based)
  - UDDI (XML Based)
- Interoperability is the key advantage of Web Services!!!

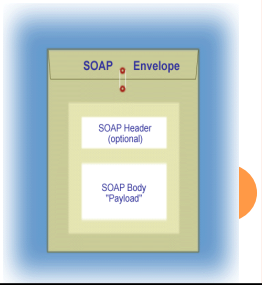
3.2.4 The Big Picture

- An application allows other applications to connect to it over the Web using SOAP
- This Web Service exposes its methods in a WSDL file
- The Web Service is published in a UDDI registry to allow other businesses to find it

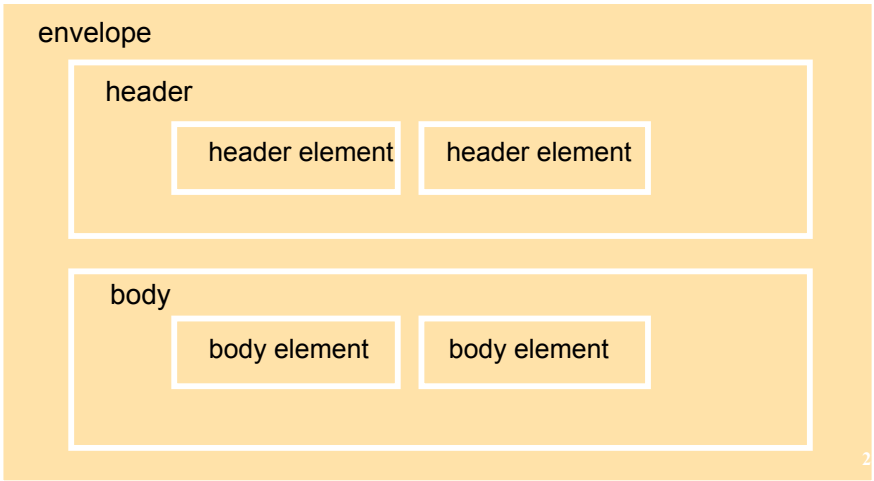
3.2.4.1 SOAP

- Simple Object Access Protocol
- SOAP is designed to enable both client-server and synchronous interaction over the Web
- Web services communicate using W3C standard SOAP messages.
- SOAP formalizes the use of XML as a way to pass data (therefore can be Objects) from one process to another.
- Originally SOAP was based only on HTTP, but other transport protocols such as TCP are also allowed.

```
<env:Envelope
xmlns:env="http://www.w3.org/2001/12/soap-envelope">
  <env:Body>
    <ns:Order xmlns:ns="urn:it.uu.se:Students">
      <item>Bill</item>
      <item>Bob</item>
      <item>Tony</item>
    </ns:Students>
  </env:Body>
</env:Envelope>
```



3.2.4.2 SOAP MESSAGES



3.2.4.2 SOAP MESSAGES

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

env:body

m:exchange
xmlns:m = namespace URI of the service description

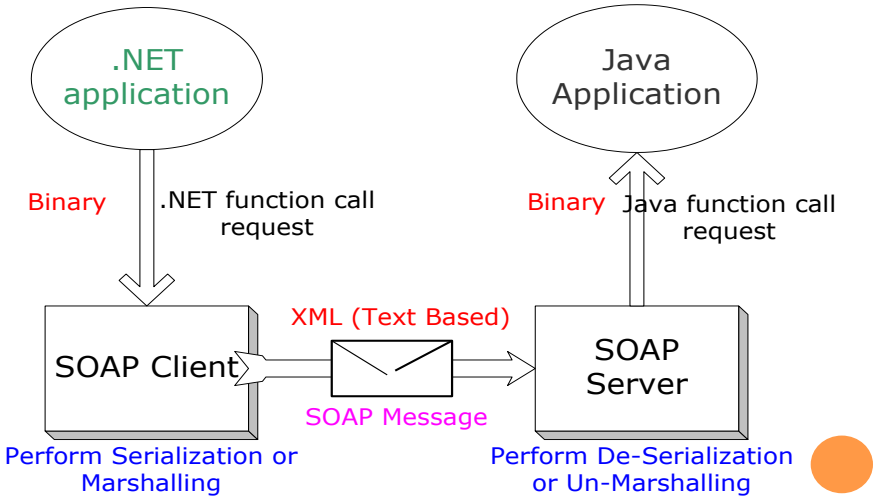
m:arg1
Hello

m:arg2
World

Example of a simple request without headers

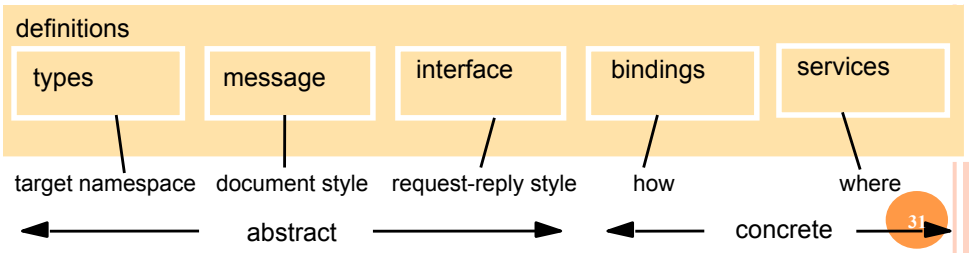
In this figure and the next, each XML element is represented by a shaded box with its name in italic, followed by any attributes and its content

3.2.4.4 TRANSPORT OF SOAP MESSAGES



3.3 Service Descriptions and IDL for Web Services

- Interface definitions are needed for clients to communicate with services.
- *Service description* specifies two characteristics – how the message are to be communicated and the URI of service.
- Web Services Description Language (WSDL)



3.3 WSDL

- WSDL has a well-defined XML vocabulary to answer
  the following questions regarding the web service involved:
- What does the service do?
  - Both in machine and human-readable forms
- What language does the service speak?
  - The format/data structure of the message exchanged
- How does the client talk to the service?
  - HTTP/SMTP/FTP
- Where is the location of the web service?
  - The access point (URL)

3.3 WSDL – MESSAGES OR OPERATIONS

- Need a common idea about the message to be exchanged

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

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

tns : target namespace xsd : XML schema definitions

WSDL request and reply messages for the newShape operation

3.3 WSDL

```
<?xml version="1.0" encoding="utf-8" ?>
<definitions xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
 xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns:s="http://www.w3.org/2001/XMLSchema"
 xmlns:s0="http://tempuri.org/"
 xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
 xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
 xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/" targetNamespace="http://tempuri.org/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
 <types>
  <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
   <s:element name="Add">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="A" type="s:float" />
      <s:element minOccurs="1" maxOccurs="1" name="B" type="s:float" />
     </s:sequence>
    </s:complexType>
   </s:element>
   <s:element name="AddResponse">
    <s:complexType>
     <s:sequence>
      <s:element minOccurs="1" maxOccurs="1" name="AddResult" type="s:float" />
     </s:sequence>
    </s:complexType>
   </s:element>
  </types>
  <message name="AddSoapIn">
   <part name="parameters" element="s0:Add" />
  </message>
  <message name="AddSoapOut">
   <part name="parameters" element="s0:AddResponse" />
  </message>
```

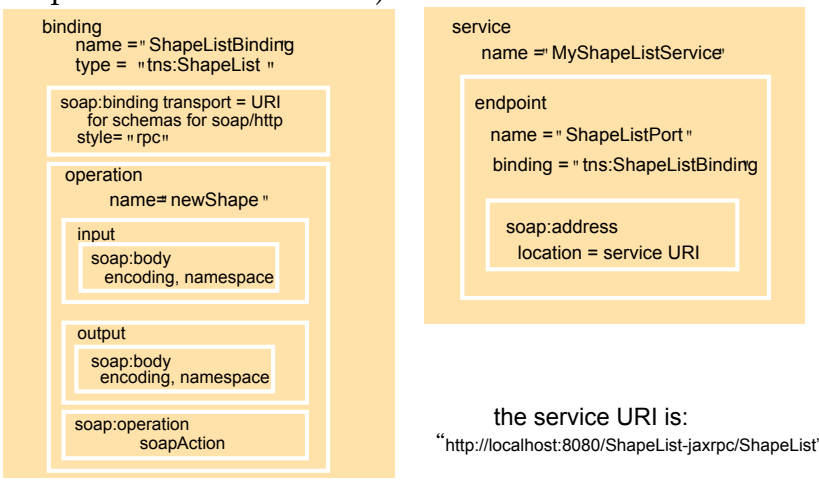
3.3 WSDL – INTERFACE

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

Message exchange patterns for WSDL operations

3.3 WSDL – CONCRETE PART

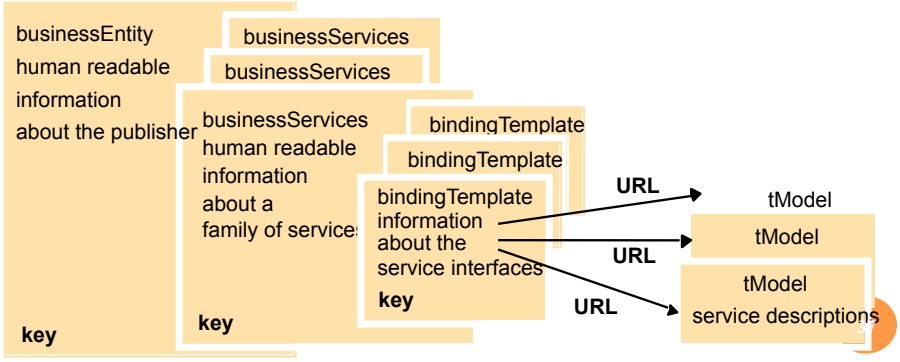
- Binding (choice of protocols) and Service (choice of endpoint or sever address):



the service URI is:
"http://localhost:8080/ShapeList-jaxrpc/ShapeList"

3.4 A DIRECTORY SERVICE FOR USE WITH WEB SERVICES

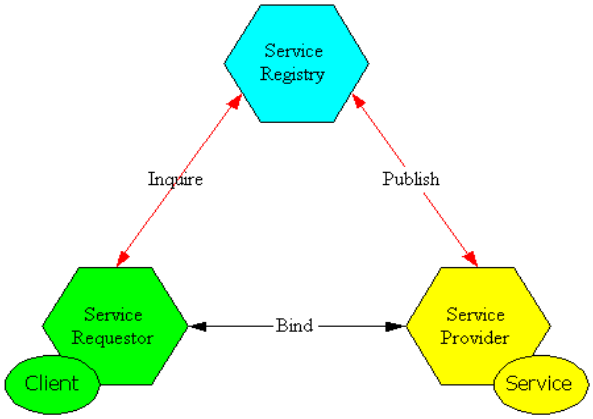
- UDDI – Universal Description, Discovery, and Integration Service
- Data structures allow human-readable information



3.4 A DIRECTORY SERVICE FOR USE WITH WEB SERVICES

- Lookup
  - UDDI provides an API for looking up services based on 2 sets of query operation: *get_xxx, find_xxx*.
  - UDDI provides a notify/subscribe interface
- Publication
  - UDDI provides an interface for publishing and updating information about web services.
- Registries
  - UDDI service is based on replicated data stored in registries

3.4 UDDI

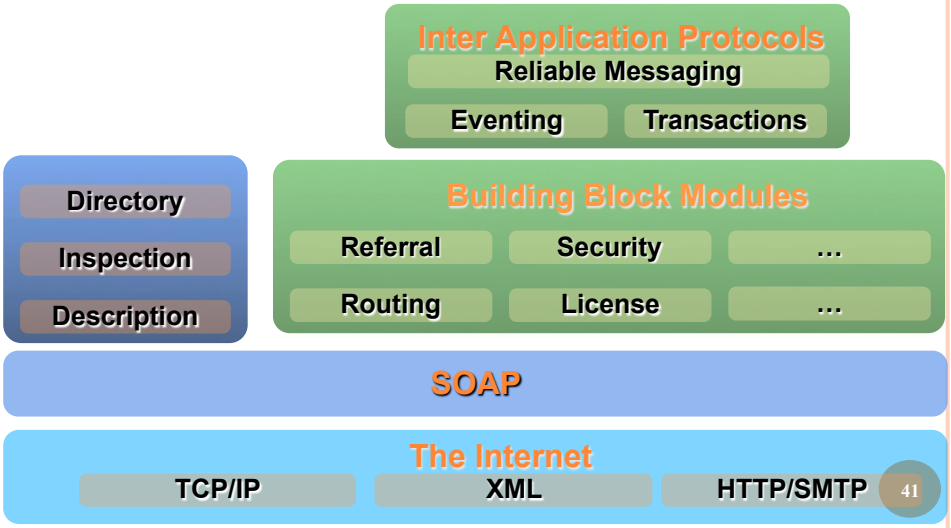


UDDI defines a way to publish and discover information about Web services

3.6 Coordination of Web Services

- SOAP infrastructure supports single request-response interactions between clients and web services.
- Web service choreography allows a set of interactions between pairs of web services working together in a joint task.
  - To generated code outline for new service that wants to participate
  - As a basis for generating test messages for a new service
  - To promote a common understanding of the collaboration
  - To analyze the collaboration, for example to identify possible deadlock situations.

3.6 Global XML Web Services Architecture



3.7 Resources of Web Services

- Web service searching engine
  - http://www.seekda.com
- Web sites
  - www.webservicex.net
  - www.xmethods.com
  - www.webservicelist.com
- Web services of well know companies:
  - Google search, Google map, Yahoo, YouTube, Facebook, Amazon, etc.

6 SUMMARY

- The basic conceptual framework for middleware in distributed systems.
- Definition of RPC and how it works.
- Basics of web services technologies.
- Read Textbook Chapters 5 and 9.

3.3 WSDL CONCRETE PART: BINDING AND SERVICE

- Binding:
  - The binding section in WSDL specifies which message formats and form of external data representation
- Service:
  - Each service element specifies the endpoints where an instance of the service may be contacted.
- Documentation:
  - Both human and machine readable information may be inserted in a documentation element at most points within a WSDL document.
- WSDL use:
  - Complete WSDL can be accessed via their URIs by clients and servers, either directly or indirectly via a directory services as UDDI.

3.3 GENERATING WSDL CODE

- Web Services generate WSDL using introspection and clients grab the WSDL file and generate code to call the service

C# Implementation Example:

```
public class MathService : WebService {

  [WebMethod]
  public float Add(float a, float b)
  {
     return a + b;
  }
}
```