Chapter 6

Service Oriented Software Architecture

Contents (Lecture 1)

- Introduction
  - Services vs. Components
  - Web Services
  - Characteristics of services
  - Example - Services scenario
  - Benefits of service-oriented approach
- Service-oriented architectures
- How to access Web services?
  - Web service Standards – SOAP Approach
  - RESTful Web Services – REST Approach
  - SOAP vs REST

Introduction

- **Service oriented development** is a means of developing distributed systems where the components are stand-alone **services**.
- A service can be defined as:
  - *A loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed. A web service is a service that is accessed using standard Internet and xML-based protocols.*
- Services may execute on different computers from different service providers.
- Standard protocols have been developed to support service communication and information exchange.
- Services are platform and implementation-language independent

Services vs. Components

- A critical distinction between a service and a component as defined in component-based software engineering is that
  - Services are independent and loosely coupled;
    - that is, they should always operate in the same way, irrespective of their execution environment.
      - Their interface is a '**provides**' interface that allows access to the service functionality. Services do not have a 'requires' interface.
        - Services are intended to be independent and usable in different contexts
  - Services rely on message-based communication with messages expressed in XML.
    - A service defines what it needs from another service by setting out its requirements in a message and sending it to that service

Services vs. Components…

- The receiving service parses the message, carries out the computation and, on completion, sends a reply, as a message, to the requesting service.
- This service then parses the reply to extract the required information
- a component is used locally (think jar file, assembly, dll, or a source import).
- A service is used remotely through some remote interface (e.g. web service, messaging system, Remote Procedure Call (RPC), or socket.)

Web services

- A web service is an instance of a more general notion of a service:

  A web service is any piece of software that makes itself available over the internet and uses a standardized xML messaging system. xML is used to encode all communications to a web service.

- The essence of a service, therefore, is that the provision of the service is independent of the application using the service.
- Service providers can develop specialized services and offer these to a range of service users from different organizations.
- A service may be as simple as "***get me some person data***," or as complex as "**process a disbursement**."
- Services provided in a SOA are deployed over the web.

Characteristics of services

- **Supports open standards for integration:**
  - Although proprietary/ownership/ integration mechanisms may be offered by the SOA infrastructure, SOA's should be based on open standards.
  - Open standards ensure the broadest integration compatibility opportunities.

- **Loose coupling:**
  - The consumer of the service is required to provide only the stated data on the interface definition, and to expect only the specified results on the interface definition.
  - The service is capable of handling all processing (including exception processing).

Characteristics of services

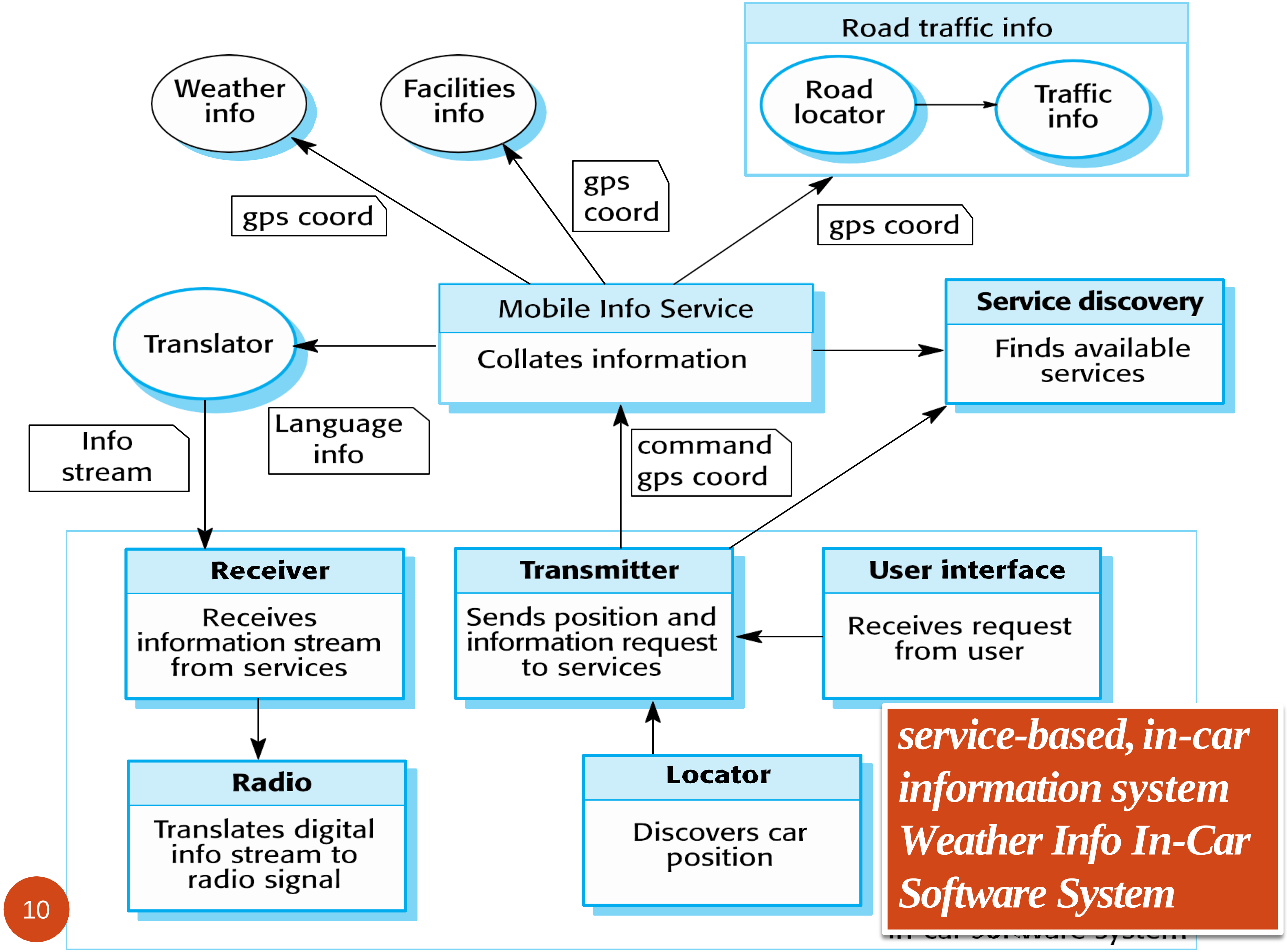
- **Stateless:**
  - The service does not maintain state between invocations.
  - It takes the parameters provided, performs the defined function, and returns the expected result.
  - If a transaction is involved, the transaction is committed and the data is saved to the database

- **Location agnostic:**
  - Users of the service do not need to worry about the implementation details for accessing the service.
  - The SOA infrastructure will provide standardized access mechanisms with service-level agreements.

Example - Services scenario

- An in-car information system provides drivers with information on weather, road traffic conditions, local information etc.
- This is linked to car audio system so that information is delivered as a signal on a specific channel.
- The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services.
- Information may be delivered in the driver’s specified language.

A service-based, in-car information system



Weather info

Facilities info

Road traffic info

Road locator

Traffic info

gps coord

gps coord

gps coord

Translator

Mobile Info Service

Collates information

Service discovery

Finds available services

Info stream

Language info

command gps coord

Receiver

Receives information stream from services

Transmitter

Sends position and information request to services

User interface

Receives request from user

Radio

Translates digital info stream to radio signal

Locator

Discovers car position

service-based, in-car information system Weather Info In-Car Software System

Benefits of service-oriented approach

- Services can be provided locally or outsourced to external providers.
- The service provider makes information about the service public so that any authorised user can use the service.
- Services are language-independent.
- Investment in legacy systems can be preserved/maintenance investment/
- Inter-organizational computing is facilitated through simplified information exchange
- Lower software development and management cost
- Ability to develop new functions rapidly

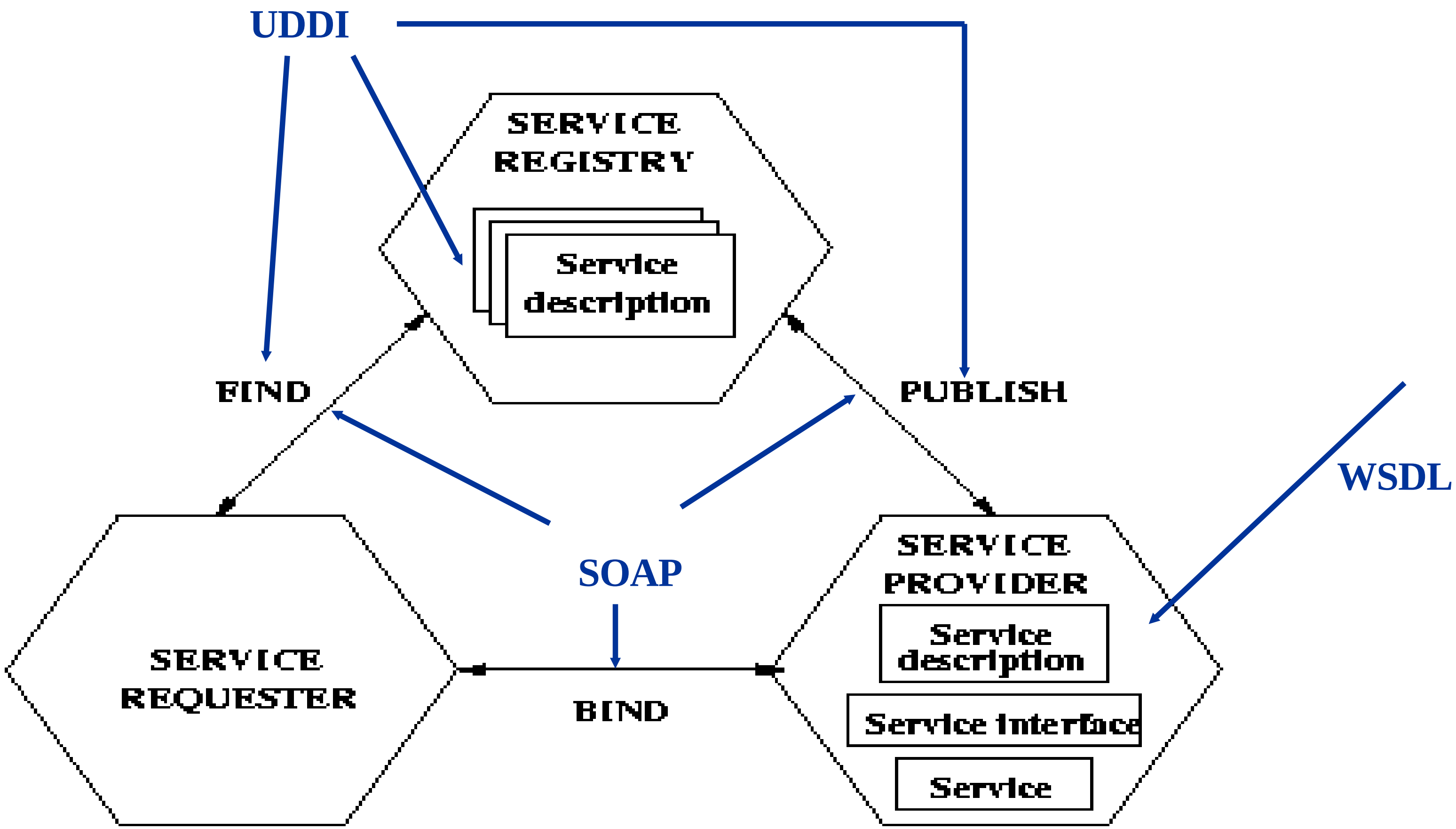
Benefits of service-oriented approach…

- Service users can pay for services according to their use rather than their provision/being provided/.
  - Instead of buying a rarely-used component, the application developers can use an external service that will be paid for only when required.
- Opportunistic construction of new services is possible.
  - A service provider may recognise new services that can be created by linking existing services in innovative ways.
- Applications can be made smaller, which is particularly important for mobile devices with limited processing and memory capabilities.
  - Computationally-intensive processing can be offloaded to external services.

Service-oriented Architecture (SOA)

- SOA is an *architectural pattern* in computer software design in which *application components* provide services to other components via a *communications protocol*, typically over a network.
  - The principles of service-orientation are independent of any vendor, product or technology.
  - Derived from the client-server architectural style.
  - Clients (service consumers or requesters) and servers (service providers) connected by a service “bus”.
  - Service bus supports point-to-point and messaging styles of communication.
  - Services defined using formal interfaces (contracts).
  - Support for system qualities, e.g., security and transaction management.

Service-oriented Architecture ...



Another architectural approach called REST can also be used to ccess Web services

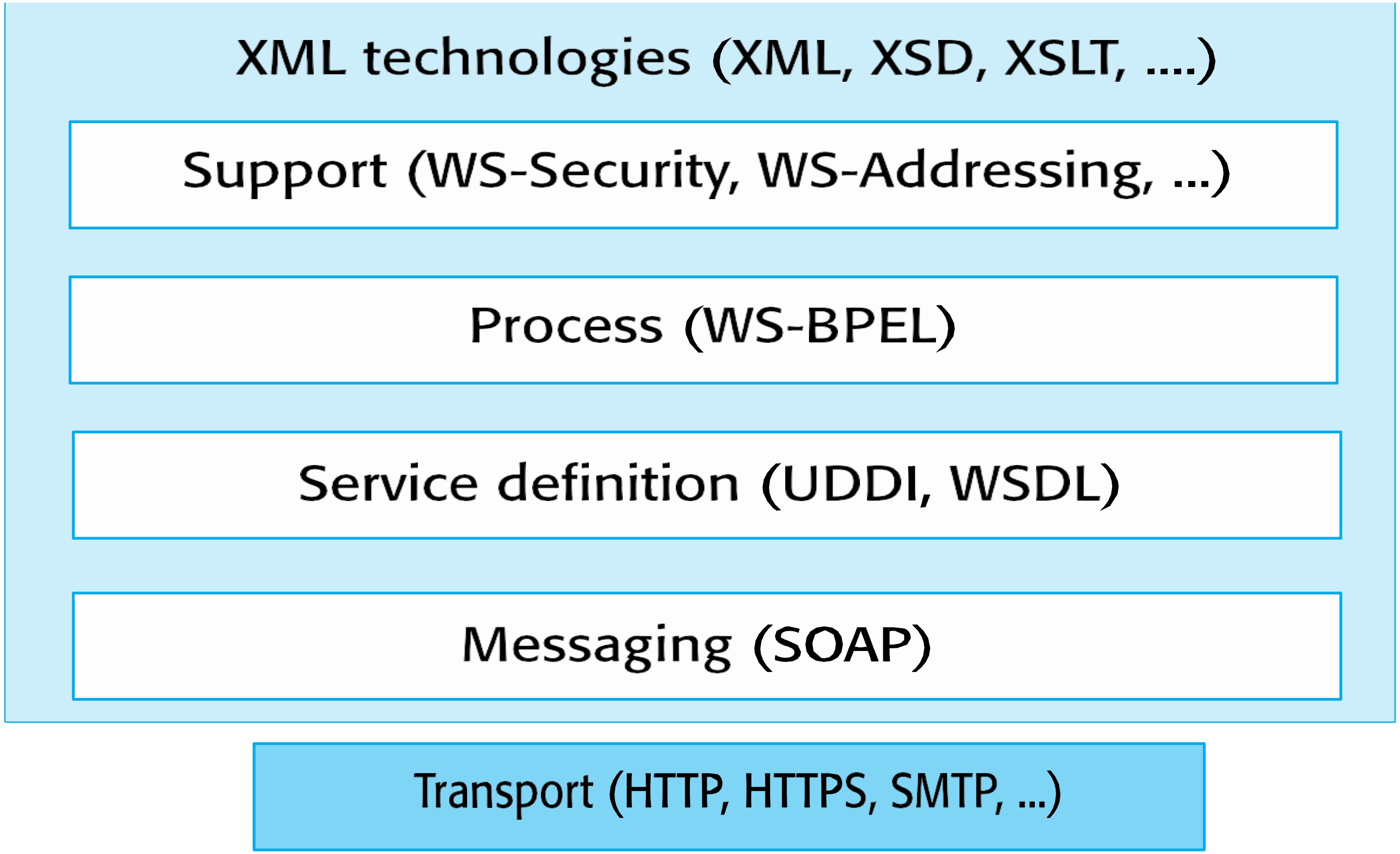
Service-oriented Architecture ...

- **Service providers**
  - design and implement services and specify the interface to these services.
  - They also publish information about these services in an accessible registry.
- **Service requestors** (sometimes called service clients)
  - who wish to make use of a service, discover the specification of that service and locate the service provider dynamically
  - They can then bind their application to that specific service and communicate with it, using standard service protocols.
- From the outset, there has been an active standardization process for SOA, working alongside technical developments.
  - All of the major HW and SW companies are committed to these standards. As a result, SOA have not suffered incompatibility problems

Web service Standards

- Web service protocols cover all aspects of SOAs, from the basic mechanisms for service information exchange (SOAP) to programming language standards (WS-BPEL).
  - ***SOAP (Simple Object Access Protocol)***
    - A message exchange standard that supports ***service communication***
  - ***WSDL (Web Service Definition Language)***
    - This standard allows a ***service interface*** and its ***bindings*** to be defined.
  - ***UDDI (Universal Description Discovery and Integration)***
    - Defines the components of a ***service specification*** that may be used to discover
  - ***WS-BPEL (Web Service Business Process Execution Language)***
    - A standard for workflow languages used to ***define service composition***
- These standards are all based on XML

Web service standards…



XML technologies (XML, XSD, XSLT, ….)

Support (WS-Security, WS-Addressing, ...)

Process (WS-BPEL)

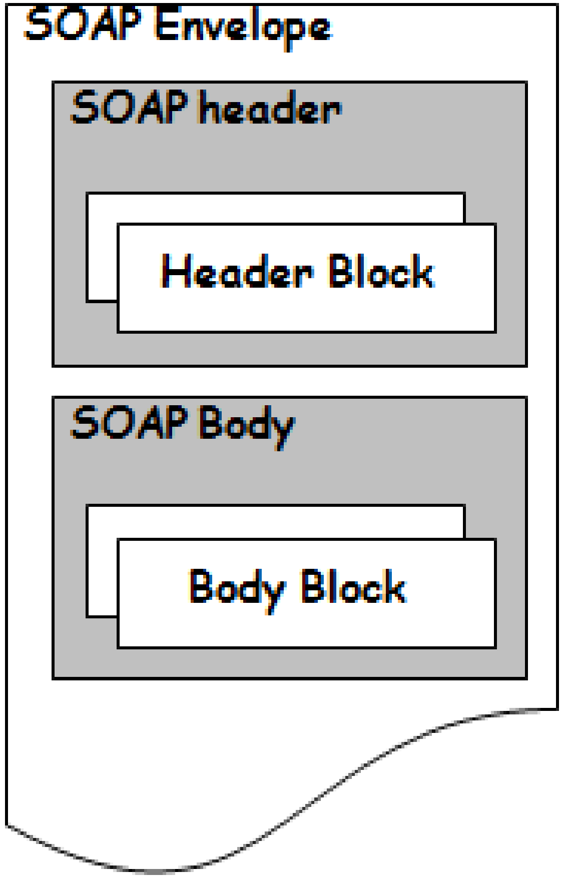
Service definition (UDDI, WSDL)

Messaging (SOAP)

Transport (HTTP, HTTPS, SMTP, ...)

SOAP

- SOAP is based on message exchanges
- Messages are seen as envelops where the application encloses the data to be sent
- A message has two main parts:
  - **header:** which can be divided into blocks
  - **body:** which can be divided into blocks
- SOAP does not say what to do with the header and the body, it only states that the header is optional and the body is mandatory
- Use of header and body, however, is implicit.
  - The body is for application level data.
  - The header is for infrastructure level data



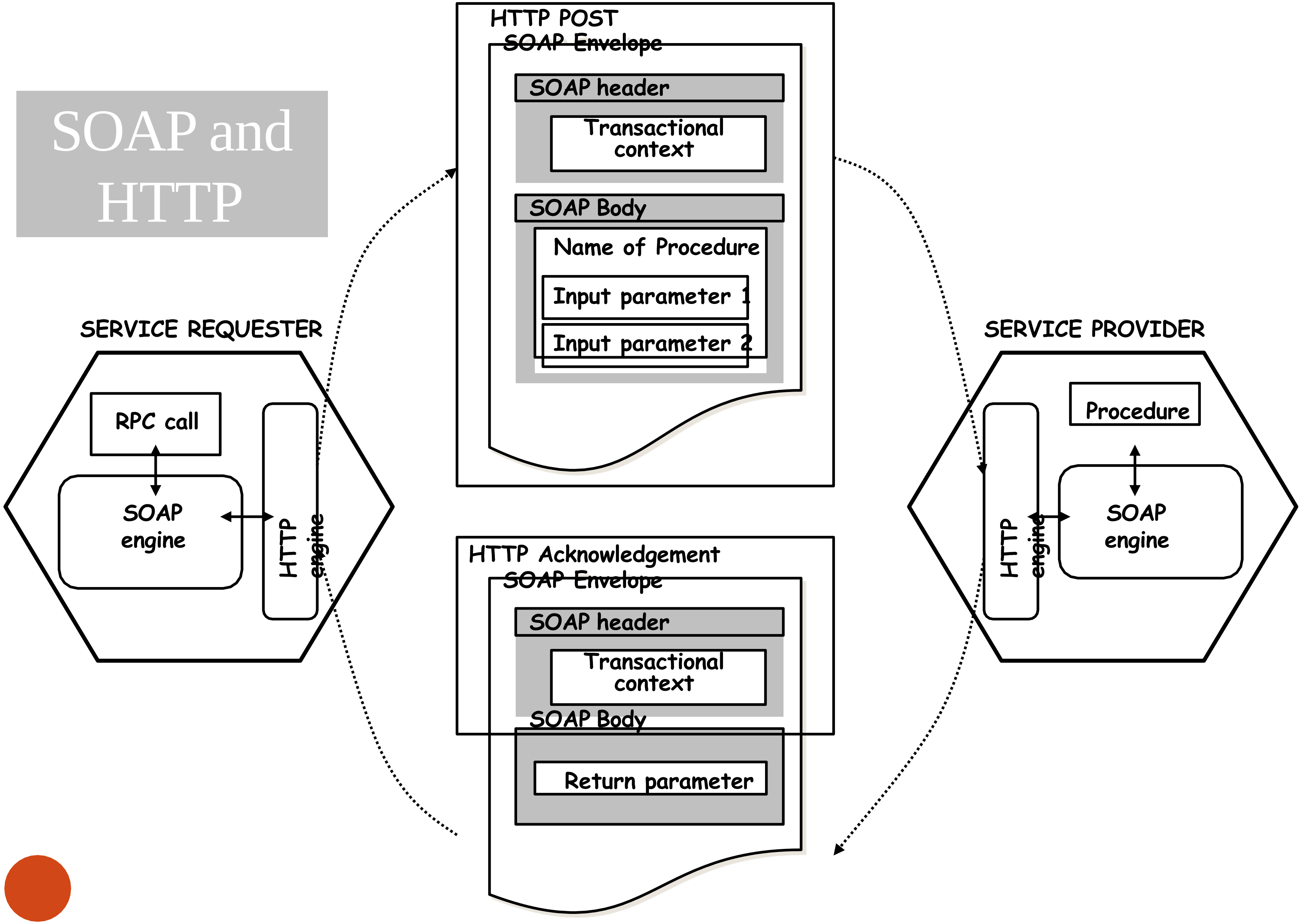
SOAP example, header and body

```
SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
     <SOAP-ENV:Header>
        <t:Transaction
           xmlns:t="some-URI"
           SOAP-ENV:mustUnderstand="1">
              5
        </t:Transaction>
     </SOAP-ENV:Header>

     <SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
           <symbol>DEF</symbol>
        </m:GetLastTradePrice>
     </SOAP-ENV:Body>

</SOAP-ENV:Envelope>
```

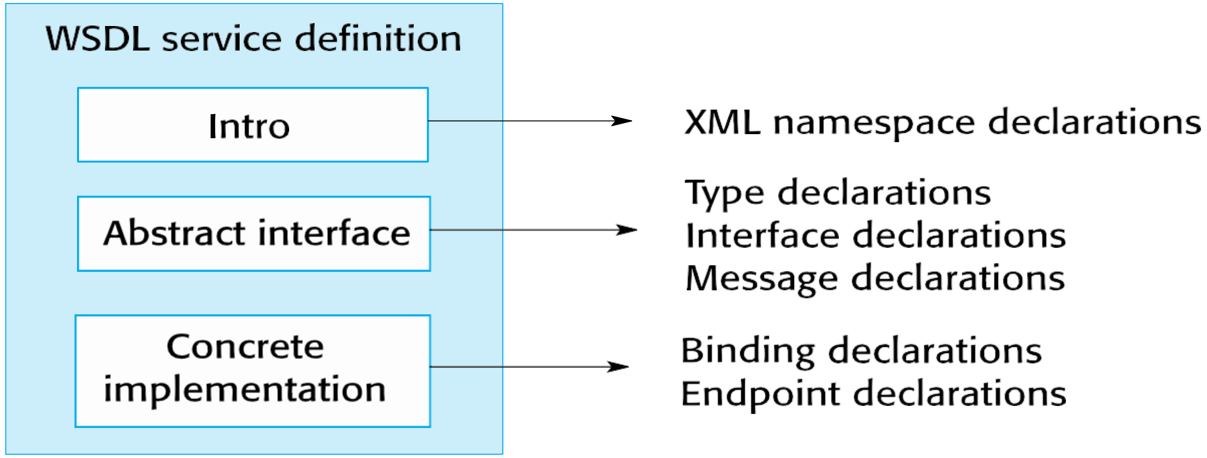
SOAP and HTTP



WSDL - Web Service Description Language

- The service interface is defined in a service description expressed in WSDL
- The WSDL specification defines
  - What ***operations*** the service supports and the format of the messages that are sent and received by the service
  - How the service is ***accessed*** - that is, the binding maps the abstract interface onto a concrete set of protocols
  - Where the service is ***located***. This is usually expressed as a URI (Universal Resource Identifier)

Organization of a WSDL specification



Part of a WSDL description for a web service

Define some of the types used. Assume that the namespace prefixes ‘ws’ refers to the namespace URI for XML schemas and the namespace prefix associated with this definition is weathns.

```
<types>
  <xs: schema targetNameSpace = “http://.../weathns”
    xmlns: weathns = “http://…/weathns” >
    <xs:element name = “PlaceAndDate” type = “pdrec” />
    <xs:element name = “MaxMinTemp” type = “mmtrec” />
    <xs: element name = “InDataFault” type = “errmess” />

    <xs: complexType name = “pdrec”
    <xs: sequence>
    <xs:element name = “town” type = “xs:string”/>
    <xs:element name = “country” type = “xs:string”/>
    <xs:element name = “day” type = “xs:date” />
    </xs:complexType>
```

Definitions of MaxMinType and InDataFault here

```
  </schema>
  </types>
```

Part of a WSDL description for a web service…

Now define the interface and its operations. In this case, there is only a single operation to return maximum and minimum temperatures.

```
<interface name = “weatherInfo” >
  <operation name = “getMaxMinTemps” pattern = “wsdlns: in-out”>
    <input messageLabel = “In” element = “weathns: PlaceAndDate” />
    <output messageLabel = “Out” element = “weathns:MaxMinTemp” />
    <outfault messageLabel = “Out” element = “weathns:InDataFault” />
  </operation>
</interface>
```

UDDI

- Deployment involves publicizing the service using UDDI and installing it on a web server.
- Current servers provide support for service installation.
- A UDDI description
  - An informal description of the functionality provided by the service.
  - Information where to find the service's WSDL specification.
  - Subscription information that allows users to register for service updates.
- **Example**
  - Consider a company XYZ wants to register its contact information, service description, and online service access information with UDDI.

UDDI…

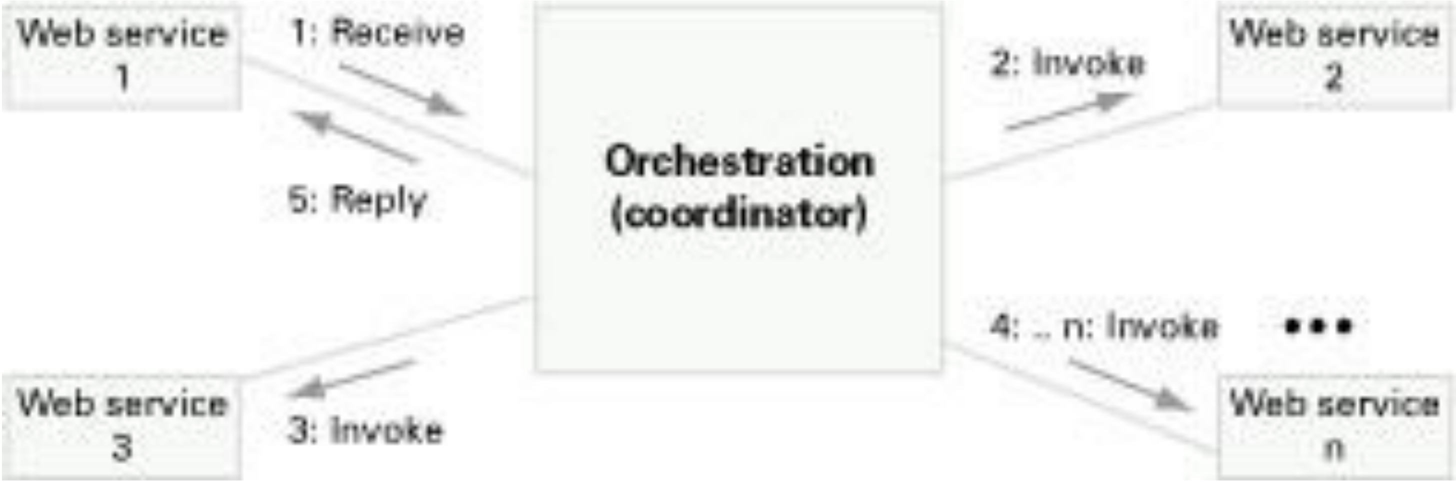
```
POST /save_business HTTP/1.1
Host: www.XYZ.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "save_business"
<?xml version="1.0" encoding="UTF-8" ?>
<Envelope xmlns="http://schemas/xmlsoap.org/soap/envelope/">
   <Body>
    <save_business generic="2.0‶ xmlns="urn:uddi-org:api_v2">
         <businessKey="">
         </businessKey>
         <name> XYZ, Pvt Ltd.</name>
         <description>Company is involved in giving Stat-of-
        the-art.... </description>
         <identifierBag> ... </identifierBag>
         ...
             </save_business>
       </Body>

        </Envelope>
```

This example illustrates a SOAP message requesting to register a UDDI business entity for XYZ Company.

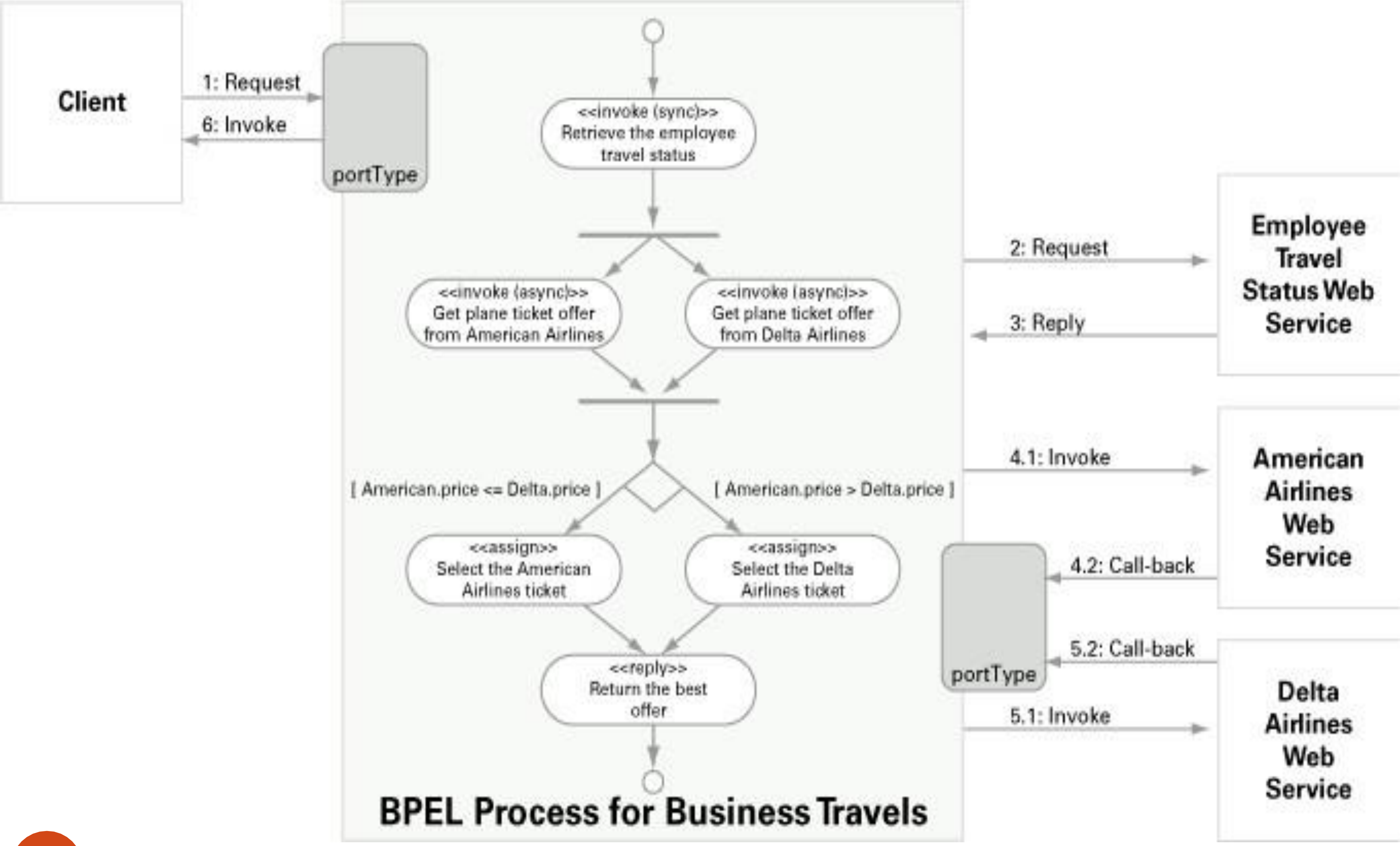
WS-BPEL

- Enables:
  - Definition of Business Processes using Web Services
  - Coordination of a set of Web service interactions
  - Degree of interoperability at the process level (describe process and use it in different runtime infrastructures)



- Where it comes from:
  - Builds on XML and Web Services
  - Convergence of two workflow languages (WSFL – directed graphs; XLANG – block-structured language)

WS-BPEL: Example


Client
1: Request
6: Invoke
portType
<<invoke (sync)>>
Retrieve the employee travel status
<<invoke (async)>>
Get plane ticket offer from American Airlines
<<invoke (async)>>
Get plane ticket offer from Delta Airlines
[American.price <= Delta.price]
[American.price > Delta.price]
<<assign>>
Select the American Airlines ticket
<<assign>>
Select the Delta Airlines ticket
<<reply>>
Return the best offer
BPEL Process for Business Travels
2: Request
3: Reply
Employee Travel Status Web Service
4.1: Invoke
4.2: Call-back
American Airlines Web Service
portType
5.2: Call-back
5.1: Invoke
Delta Airlines Web Service

Composite Service - Orchestrated by BPEL

Structure of a BPEL4WS Process

```
<process ...>

   <partners> ... </partners>
      <!--Web services the process interacts with -->
   <containers> ... </containers>
      <!– Data used by the process -->
   <correlationSets> ... </correlationSets>
      <!– Used to support asynchronous interactions -->
   <faultHandlers> ... </faultHandlers>
      <!–Alternate execution path to deal with faulty conditions -->
   <compensationHandlers> ... </compensationHandlers>
                           <!–Code to execute when “undoing” an action -->
   (process body)
      <!– What the process actually does -->

</process>
```

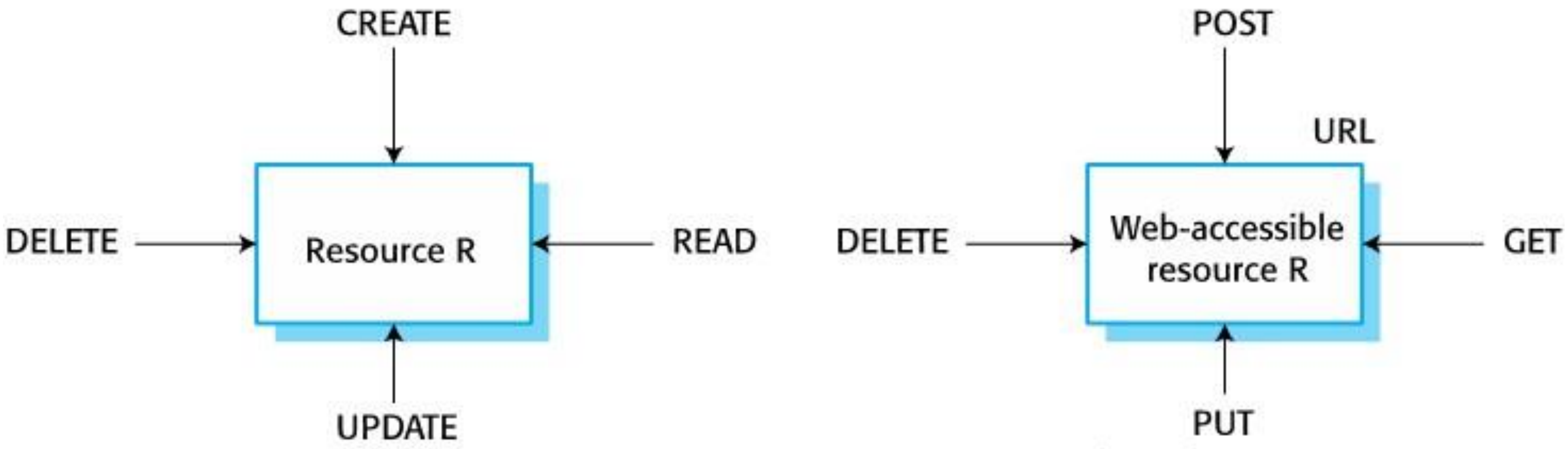
RESTful Web Services

- Current web services standards have been criticized as ‘heavyweight’ standards that are over-general and inefficient.
- RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web based applications.
- REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client.
- This style underlies the web as a whole and is simpler than SOAP/WSDL for implementing web services.
- RESTful services involve a lower overhead and are used by many organizations implementing service-based systems.

Resources

- The fundamental element in a RESTful architecture is a resource.
- Essentially, a resource is simply a data element such as a catalog, a medical record, or a document
- In general, resources may have multiple representations i.e. they can exist in different formats.
  - MSWORD, PDF, Quark Xpress
- Resource operations
  - Create – bring the resource into existence.
  - Read – return a representation of the resource.
  - Update – change the value of the resource.
  - Delete – make the resource inaccessible.

Resources and actions



(a) General resource actions

(b) Web resources

- Operation functionality
  - POST is used to create a resource. It has associated data that defines the resource.
  - GET is used to read the value of a resource and return that to the requestor in the specified representation, such as XHTML, that can be rendered in a web browser.
  - PUT is used to update the value of a resource.
  - DELETE is used to delete the resource.

Resource access

- When a RESTful approach is used, the data is exposed and is accessed using its URL.
- Therefore, the weather data for each place in the database, might be accessed using URLs such as:
  - http://weather-info-example.net/temperatures/boston http://weather-info-example.net/temperatures/edinburgh
- Service requester invokes the GET operation and returns a list of maximum and minimum temperatures.
- To request the temperatures for a specific date, a URL query is used:
  - http://weather-info-example.net/temperatures/edinburgh?date=20140226

Query results

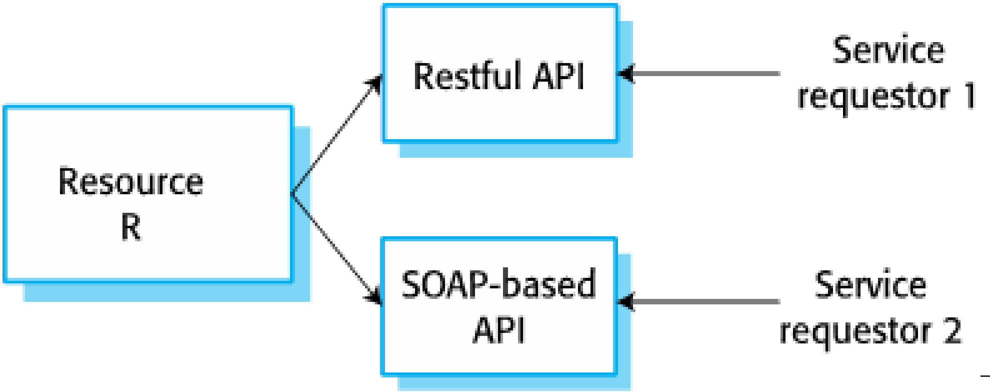
- The response to a GET request in a RESTful service may include URLs.
- If the response to a request is a set of resources, then the URL of each of these may be included.
  - http://weather-info-example.net/temperatures/edinburgh-scotland
  - http://weather-info-example.net/temperatures/edinburgh-australia
- **Disadvantages** of RESTful approach
  - It can be difficult to design a set of RESTful services to represent complex interface
  - no standards for RESTful interface description so that users
    - must rely on informal documentation to understand the interface.
    - have to implement your own infrastructure for monitoring and managing the quality of service and the service reliability.

SOAP vs REST

- SOAP is definitely the heavyweight choice for Web service access.
- However, it provides the following advantages when compared to REST:
  - Language, platform, and transport independent (REST requires use of HTTP)
  - Works well in distributed enterprise environments (REST assumes direct point-to-point communication)
  - Standardized
  - Provides significant pre-build extensibility in the form of the WS* standards
  - Built-in error handling
  - Automation when used with certain language products

SOAP vs REST…

- REST is easier to use for the most part and is more flexible.
- And it has the following **advantages** when compared to SOAP:
  - No expensive tools require to interact with the Web service
  - Smaller learning curve
  - Efficient (SOAP uses XML for all messages, REST can use smaller message formats)
  - Fast (no extensive processing required)
  - Closer to other Web technologies in design philosophy
- SOAP and REST are not incompatible and it is possible for web services providers to offer both SOAP and REST service interfaces.


Restful API
Service requestor 1
Resource R
SOAP-based API
Service requestor 2