Chapter 6

Service Oriented Software Architecture

By Yohannes S.

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

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

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.)

- For more follow the following link

https://ifs.host.cs.st-andrews.ac.uk/Books/SE9/Web/Services/Comps.html

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.

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

Characteristics of services

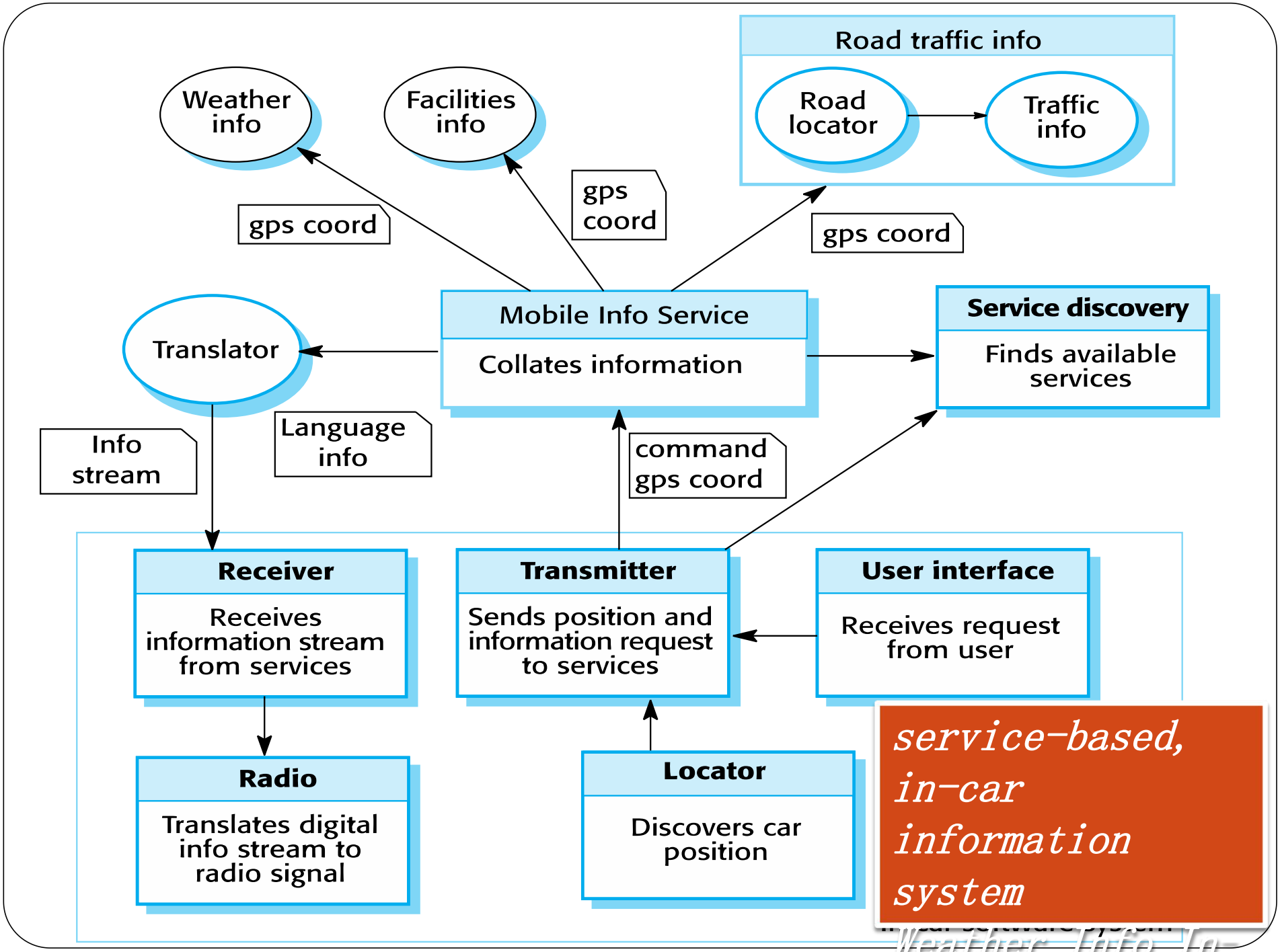
- Stateless:
  - The service does not maintain state between invocations.
  - It takes the parameters provided, performs the defined function, and returns the expected result.
  - If a transaction is involved, the transaction is committed and the data is saved to the database

- 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

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 drive

A service-based, in-car information system


Road traffic info
Road locator
Traffic info
Weather info
Facilities info
gps coord
gps coord
gps coord
Mobile Info Service
Collates information
Translator
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

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

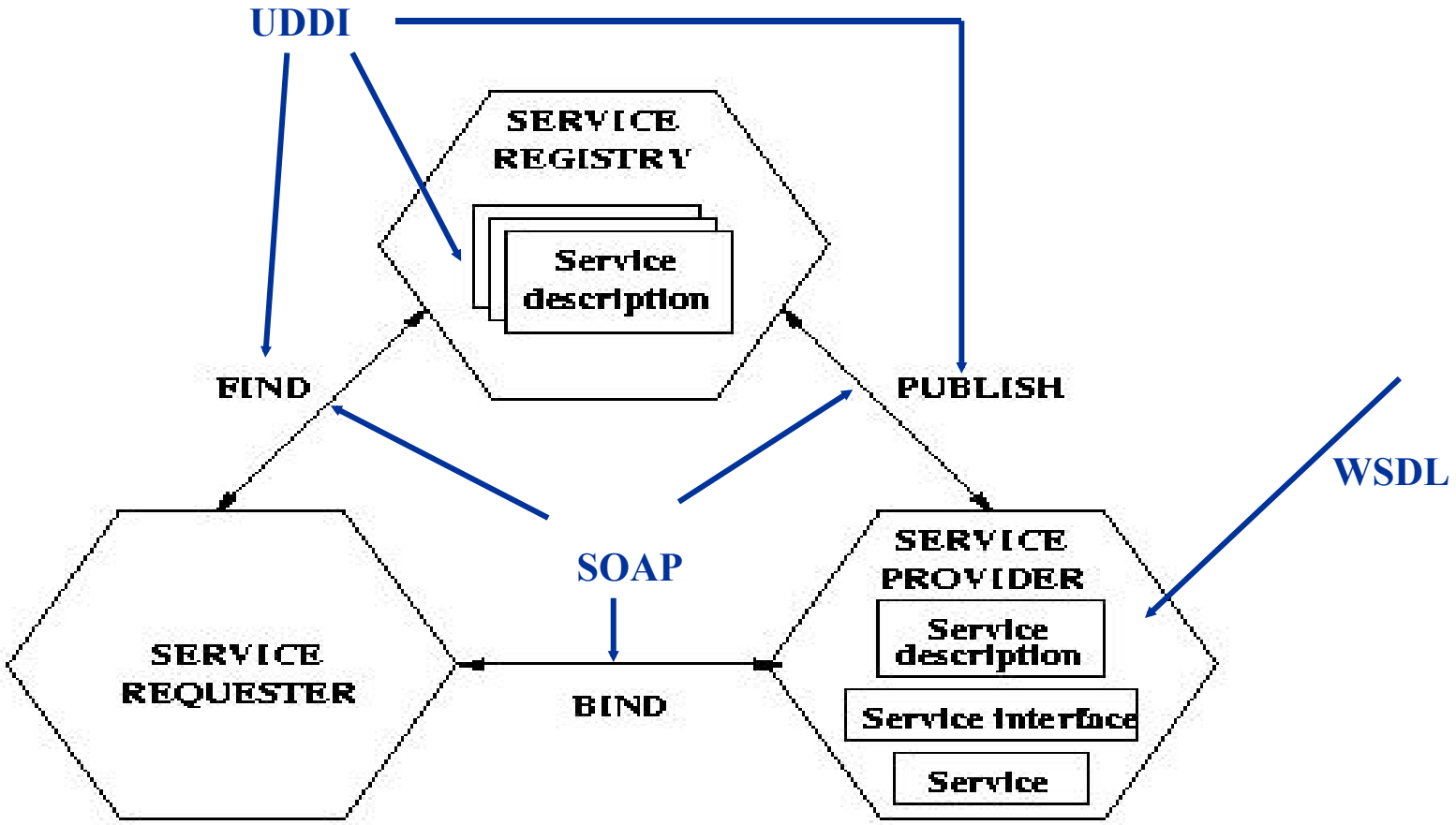
Benefits of service oriented approach…

- 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

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.

Service Oriented Architecture ...



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

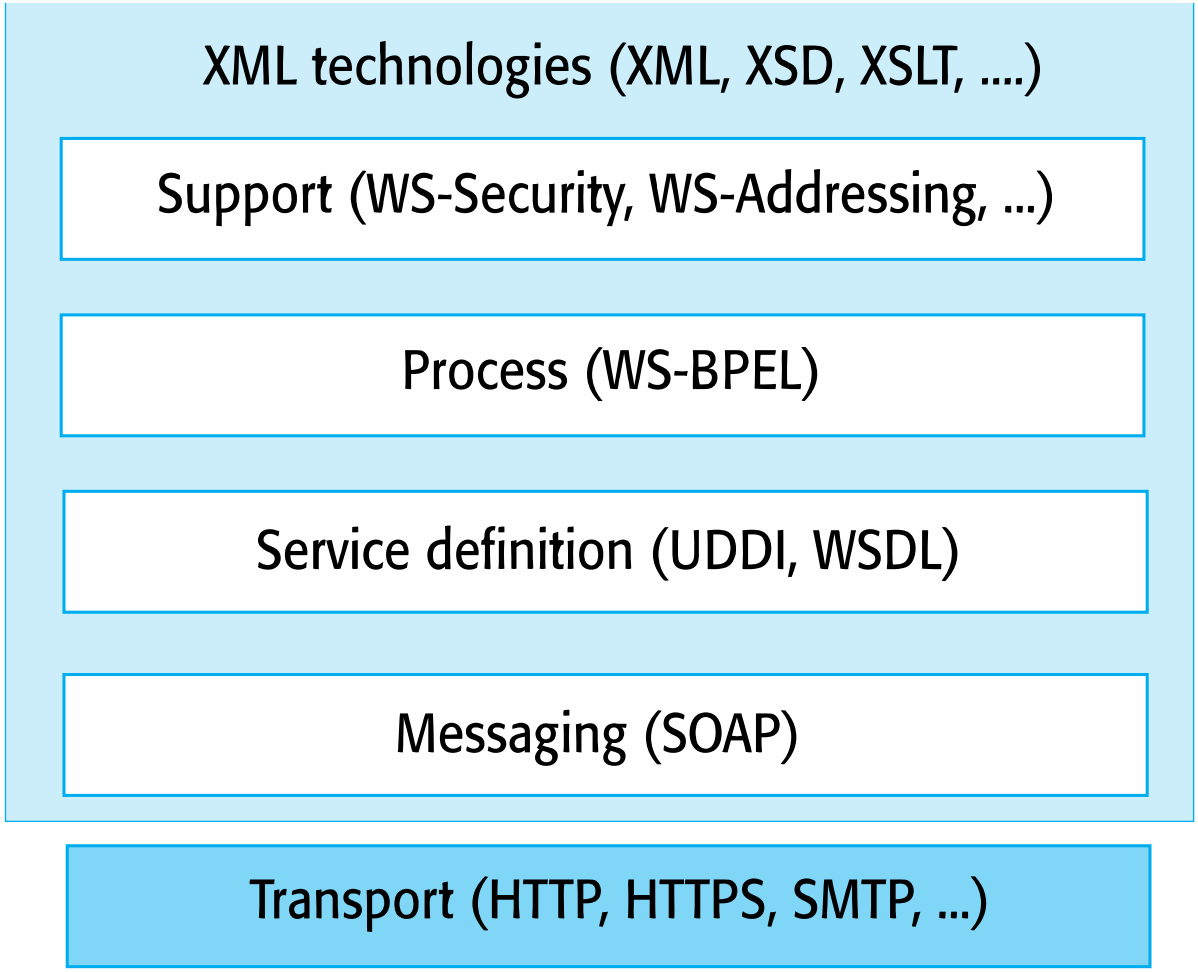
Service Oriented Architecture ...

- **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

Web service Standards

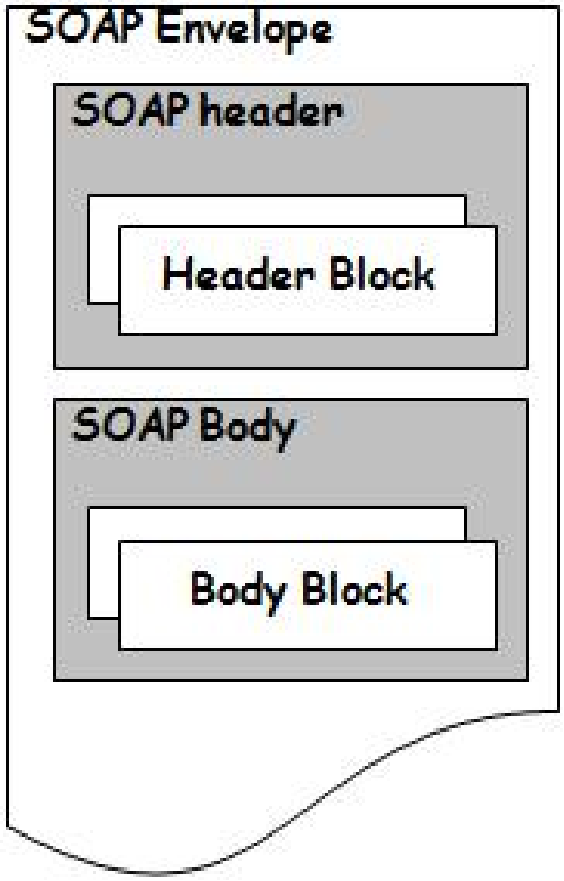
- Web service protocols cover all aspects of SOAs, from the basic mechanisms for service information exchange (SOAP) to programming language standards (WS-BPEL).
  - *SOAP (Simple Object Access Protocol)*
    - A message exchange standard that supports *service communication*
  - *WSDL (Web Service Definition 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*

Web service standards…



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.



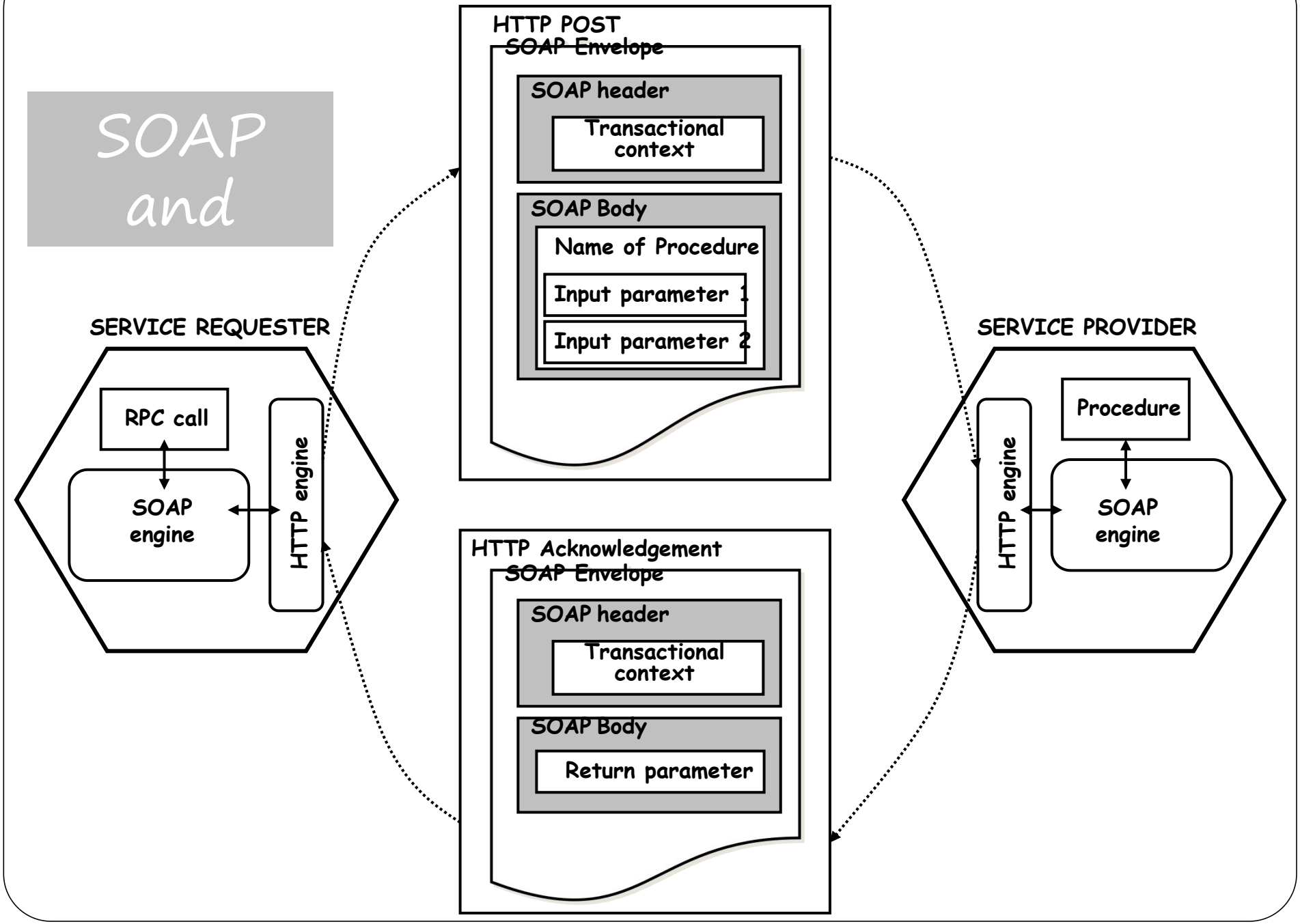
SOAP example, header and body

```
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>
```

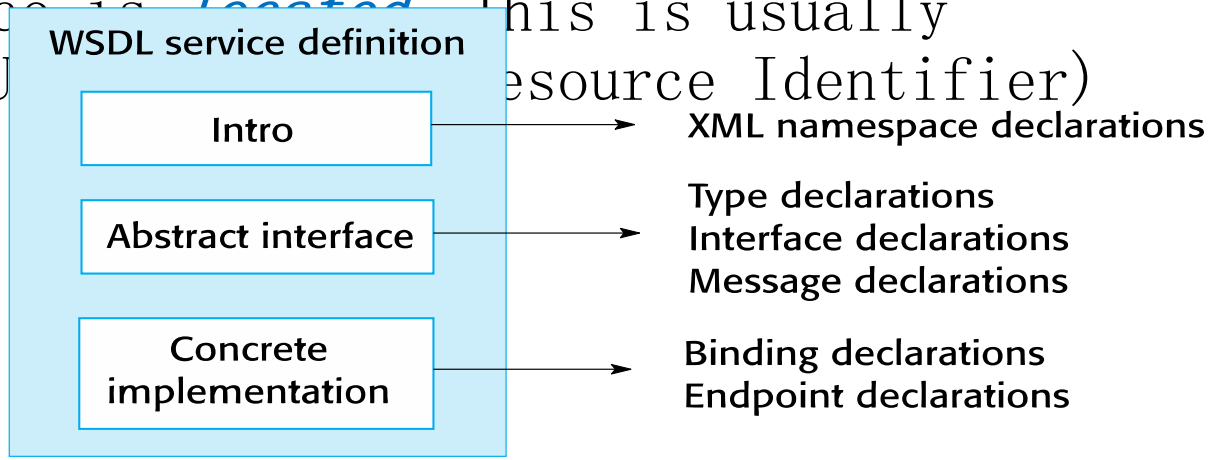
SOAP and


HTTP POST
SOAP Envelope
SOAP header
Transactional context
SOAP Body
Name of Procedure
Input parameter 1
Input parameter 2
SERVICE REQUESTER
RPC call
SOAP engine
HTTP engine
SERVICE PROVIDER
Procedure
SOAP engine
HTTP engine
HTTP Acknowledgement
SOAP Envelope
SOAP header
Transactional context
SOAP Body
Return parameter

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 U esource 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

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>
```

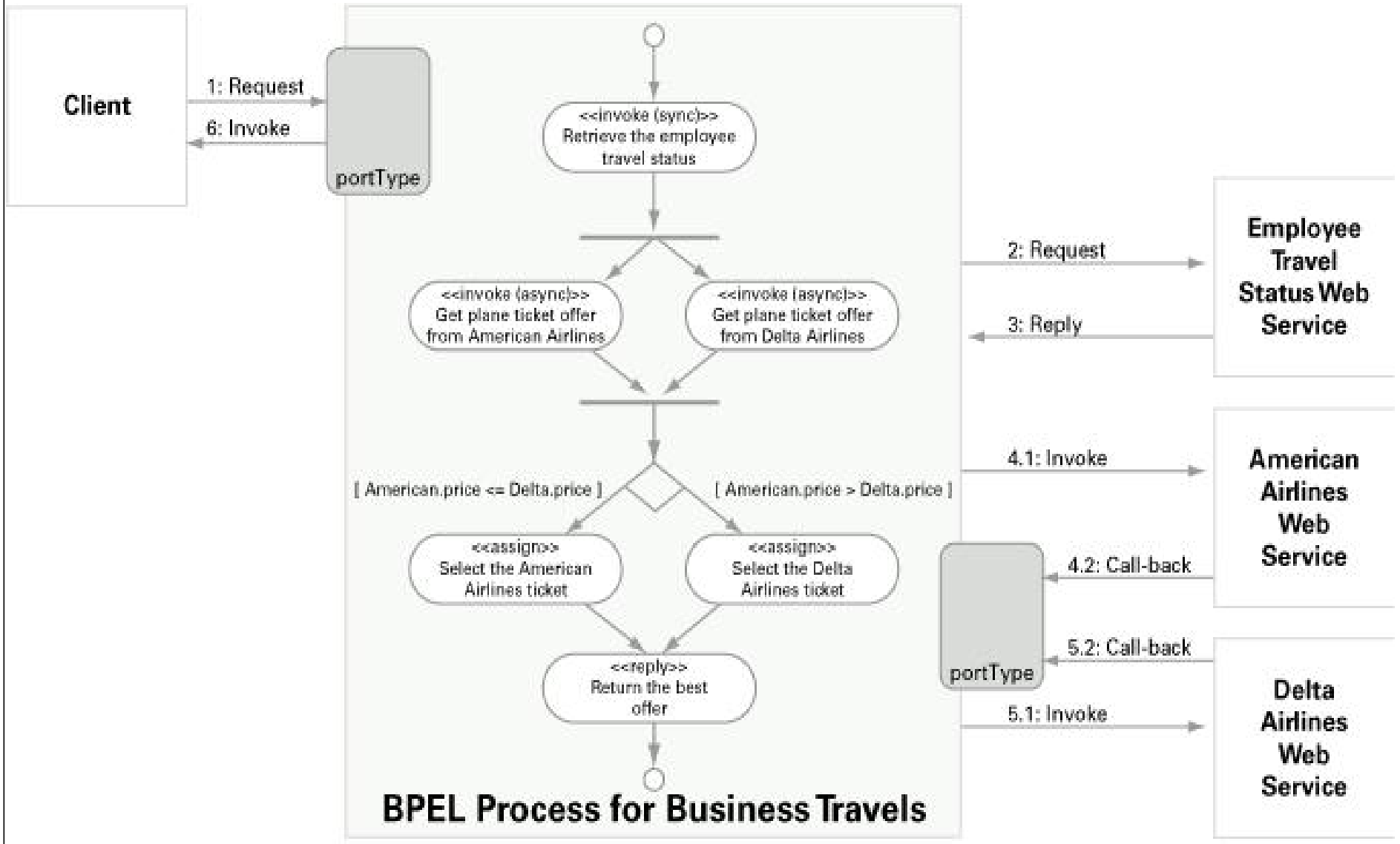
This example illustrates a SOAP message requesting to register a UDDI business entity for XYZ Company

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

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

WS-BPEL: Example



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>
```

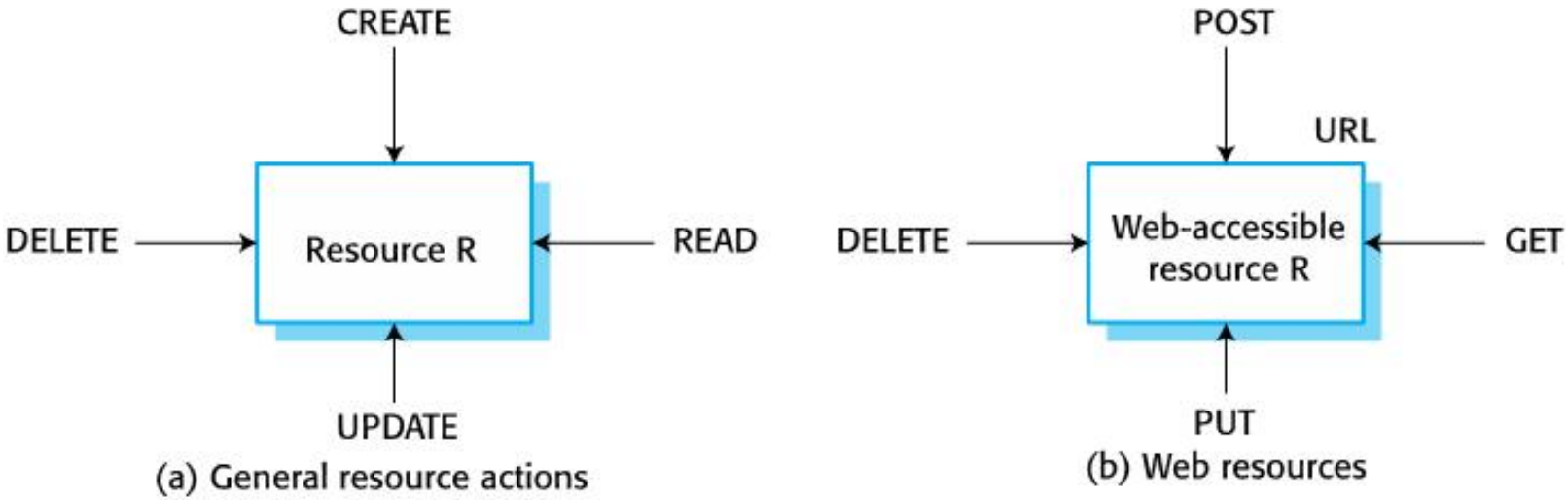
RESTful Web Services

- 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

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.
  - MS WORD, 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.

Resources and actions



- 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=

Query results

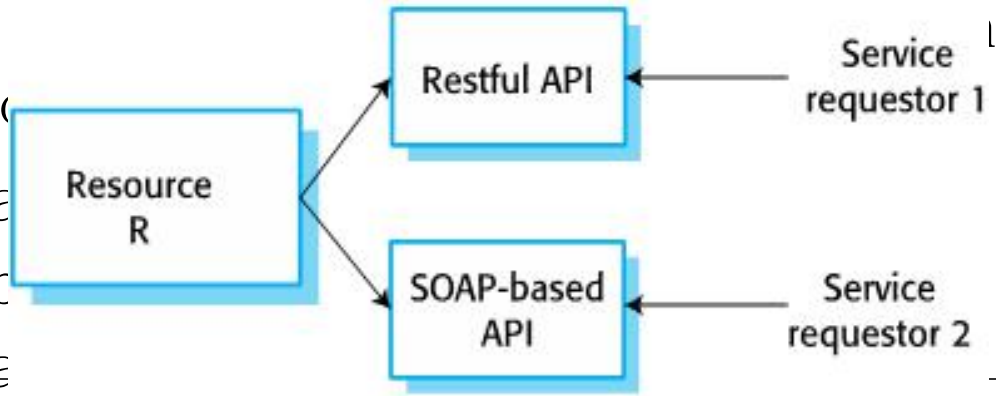
- The response to a GET request in a RESTful service may include URLs.
- If the response to a request is a set of resources, then the URL of each of these may be included.
  - http://weather-info-example.net/temperatures/edinburgh-scotland
  - http://weather-info-example.net/temperatures/edinburgh-australia
- **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

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)
  - Clos… design phil…
- SOAP a… and it is possib… s to offer both SOAP a…



Contents (Lecture 2)

- Service-Oriented Software Engineering
- Service engineering
  - Service Engineering Process
    - Service candidate identification
    - Service design
    - Service implementation and deployment
  - Legacy system services
- Software development with services
  - Workflow design and implementation
  - Service testing
- Summary

Service Oriented Software Engineering

- is a software engineering methodology focused on the development of software systems by composition of reusable services often provided by other service providers.
  - Services are from various providers using either a standard programming language or a specialized workflow language
- Since it involves composition, it shares many characteristics of component-based software engineering, the composition of software systems from reusable components,
  - but it adds the ability to dynamically locate necessary services at run-time
- It is as significant a development as

Service Oriented Software Engineering

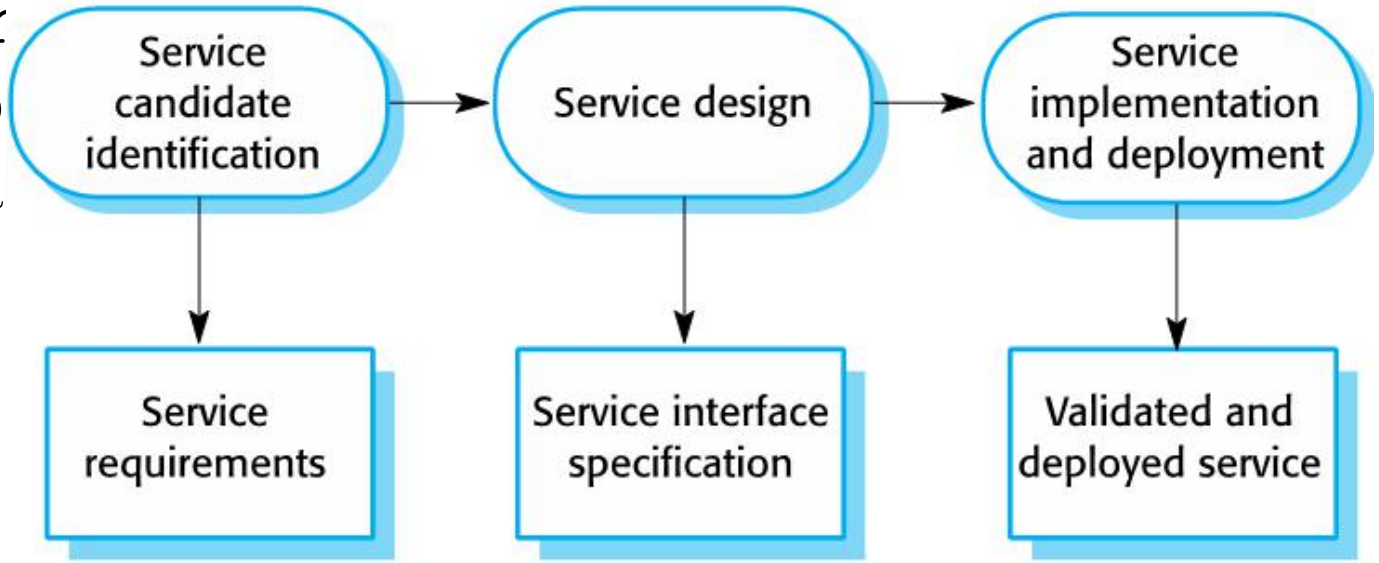
- Building applications based on services allows companies and other organizations to cooperate and make use of each other's business functions.
- Existing approaches to software engineering have to evolve to reflect the service-oriented approach to software development
- Service-Oriented Software Engineering contains
  - **Service Engineering**
    - The development of dependable, reusable services – *Software development for reuse*
  - **Software Development with Services**
    - The development of dependable software where

Service Engineering

- Is the process of developing services for reuse in service-oriented applications
- The service has to be designed as a **reusable abstraction** that can be used in different systems
- Generally useful functionality associated with that abstraction must be designed and the service must be **robust** and **reliable.**
- The service must be **documented** so that it can be discovered and understood by potential users.
- The starting point for service engineering

The Service Engineering Process

- Stages of service engineering
  - **Service candidate identification** - identify possible services that might be implemented and define the service requirements.
  - **Service design** - design the logical service interface and its implementation interfaces (SOAP and/or RESTful)
  - **Ser**
    imp
    ava



Service candidate identification

- Services should support business processes.
- Service candidate identification involves understanding an organization's business processes to decide which reusable services could support these processes.
- Three fundamental types of service
  - **Utility services** that implement general functionality used by different business processes. E.g. currency conversion service [e.g., dollars to another euros]
  - **Business services** that are associated with a specific business function e.g., in a university, student registration.
  - **Coordination services** that support composite

Service candidate identification…

- Task and entity-oriented services
  - Services can also be thought of as task-oriented or entity oriented.
  - Task-oriented services are those associated with some activity.
  - Entity-oriented services are like objects. They are associated with a business entity such as a job application form

	Utility	Business	Coordination
Task	Currency converter Employee locator	Validate claim form Check credit rating	Process expense claim Pay external supplier
Entity	Document style checker , Web	Expenses form, Student	

Service candidate identification…

- The goal of service candidate identification should be to identify services that are logically coherent, independent, and reusable.
- Preceding classification is helpful in this respect as it suggests how to discover reusable services by looking at business entities and business activities.
- However, identifying service candidates is sometimes difficult because you have to envisage /imagine possibility/ how the services will be used.
- You have to think of possible candidates then ask a series of questions about them to see if

Service candidate identification…

- Is the service associated with a single logical **entity** used in different business processes?
  - What operations are normally performed on that entity that must be supported?
- Is the **task** one that is carried out by different people in the organization? Can this fit with a RESTful model?
- Is the service independent (i.e., to what extent does it rely on the availability of other services)?
- Does the service have to maintain ***state***? Is a ***database*** required?
- Could the service be used by clients outside the organization?

Service Identification example

- *A large company, which sells computer equipment, has arranged special prices for approved configurations for some customers.*
- *To facilitate automated ordering, the company wishes to produce a catalog service that will allow customers to select the equipment that they need.*
- *Unlike a consumer catalog, orders are not placed directly through a catalog interface. Instead, goods are ordered through the web-based procurement system of each company that accesses the catalog as a web service.*
- *Most companies have their own budgeting and*

Service identification example…

- Catalog services
  - Created by a supplier to show which good can be ordered from them by other companies
    - *It is an example of an entity-oriented service that supports business operations.*
  - The functional catalog service requirements are as follows:

1. *Specific version of catalogue should be created for each client*
2. *Catalogue shall be downloadable*
3. *The specification and prices of up to 6 items may be compared*
4. *Browsing and searching facilities shall be provided*

Service identification example…

- Catalogue: Non-functional requirements
  1. *Access shall be restricted to employees of accredited organizations*
  2. *Prices and configurations offered to each organization shall be confidential*
  3. *The catalogue shall be available from 0700 to 1100*
  4. *The catalogue shall be able to process up to 10 requests per second*

Notice that there is no non-functional requirement related to the response time of the catalog service. This depends on the size of the

Service interface design

- Involves thinking about the operations associated with the service and the messages exchanged
- The number of messages exchanged to complete a service request should normally be minimized.
- Service state information may have to be included in messages
- **Interface design stages**
  1. **Logical interface design**
     - Starts with the service requirements and defines the operation names and parameters associated with the service.
     - Exceptions that may arise when a service

Service interface design …

Functional descriptions of catalog service operations

Operation	Description
MakeCatalog	Creates a version of the catalog tailored for a specific customer. Includes an optional parameter to create a downloadable PDF version of the catalog.
Lookup	Displays all of the data associated with a specified catalog item.
Search	This operation takes a logical expression and searches the catalog according to that expression. It displays a list of all items that match the search expression.
Compare	Provides a comparison of up to six characteristics (e.g., price, dimensions, processor speed, etc.) of up to four catalog items.

Operation	Inputs	Outputs	Exceptions
MakeCatalog	*mcIn* Company id PDF-flag	*mcOut* URL of the catalog for that company	*mcFault* Invalid company id
Compare	*compIn* Company id Entry attribute (up to 6) Catalog number (up to 4)	*compOut* URL of page showing comparison table	*compFault* Invalid company id Invalid catalog number Unknown attribute
Lookup	*lookIn* Company id Catalog number	*lookOut* URL of page with the item information	*lookFault* Invalid company id Invalid catalog number
Search	*searchIn* Company id Search string	*searchOut* URL of web page with search results	*searchFault* Invalid company id Badly formed search string
CheckDelivery	*gdIn* Company id Catalog number Number of items required	*gdOut* Catalog number Expected delivery date	*gdFault* Invalid company id Invalid catalog number No availability Zero items requested
PlaceOrder	*poIn* Company id Number of items required Catalog number	*poOut* Catalog number Number of items required Predicted delivery date Unit price estimate Total price estimate	*poFault* Invalid company id Invalid catalog number Zero items requested

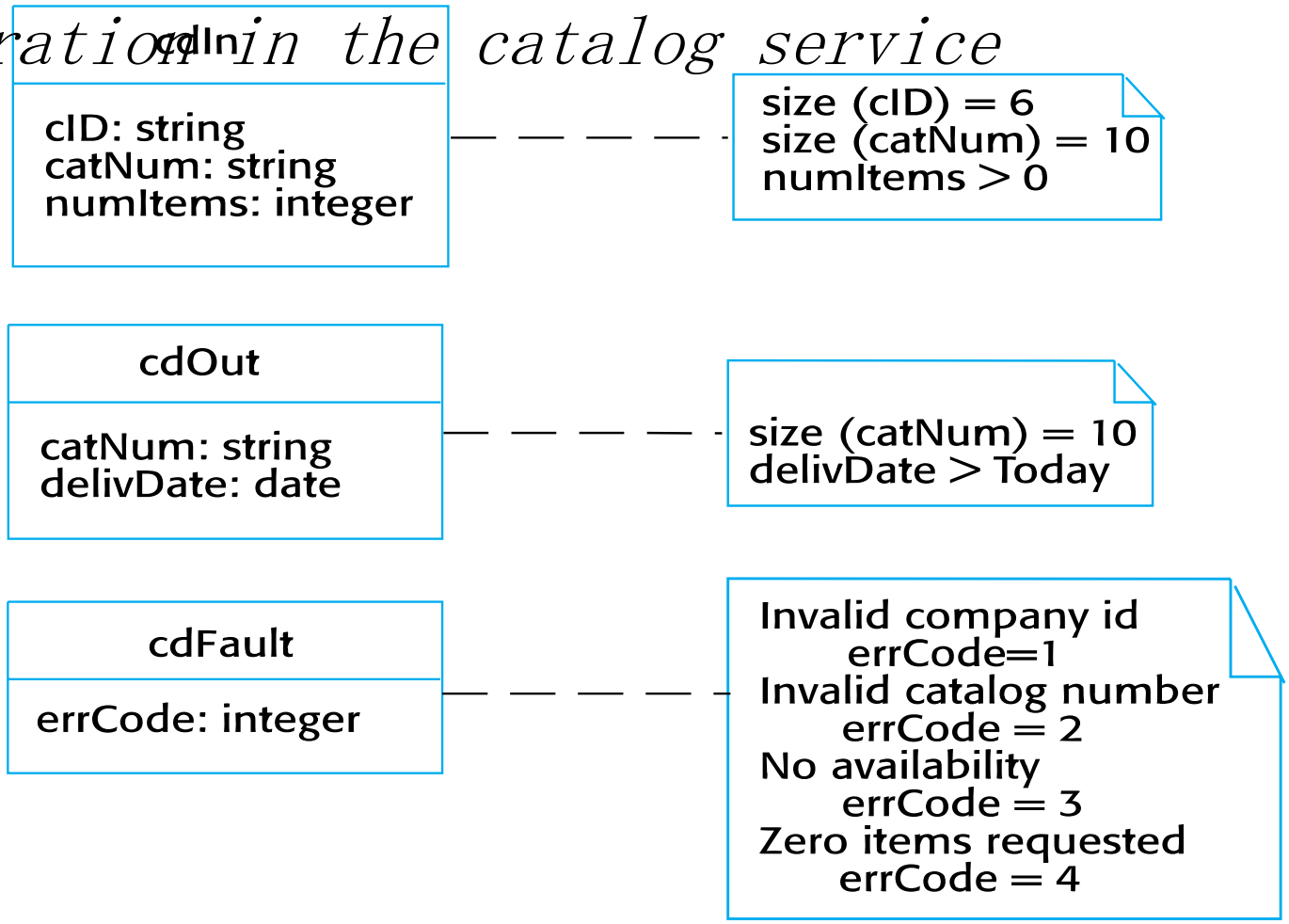
UML definition of input & output

Service interface design …

2. Message design (SOAP)
   - where you design the structure of the messages that are sent and received by the service.
   - For SOAP-based services, design the structure and organization of the input and output messages.
   - Notations such as the UML are a more abstract representation than XML
     - It is better to represent the messages as objects and either define them using the UML or in a programming language, such as Java
     - They can then be manually or automatically

Service interface design …

The diagram below shows the structure of the input and output messages for the ***getDelivery*** *operation in the catalog service*



Service interface design …

3. **WSDL development,**
   - where you translate your logical and message design to an abstract interface description written in WSDL
   - The logical specification is converted to a WSDL description
     - Most programming environments that support service-oriented development (e.g., the ECLIPSE environment) include tools that can translate a logical interface description into its corresponding WSDL representation
       - WSDL representation is long and detailed and hence it is easy to make mistakes at this stage if you do this manually.

Service identification example…

- RESTful interface
  - In case of REST, design how the required operations map onto REST operations and what resources are required.
  - There should be a resource representing a company-specific **catalog**
  - This should have a URL of the form *<base catalog>/<company name>* and should be created using a **POST** operation.
  - Each catalog item should have its own URL of the form:
    - *<base catalog>/<company name>/<item identifier>.*

Service identification example…

- **Search** is implemented by using **GET** with the company catalog as the URL and the search string as a query parameter. This GET operation returns a list of URLs of the items matching the search.
- The **Compare** operation can be implemented as a sequence of GET operations, to retrieve the individual items, followed by a POST operation to create the comparison table and a final GET operation to return this to the user.
- The **CheckDelivery** and **MakeVirtualOrder** operations require an additional resource, representing a virtual order.
- A POST operation is used to create this

Service implementation and deployment

- The final stage of the service engineering process is service implementation.
- This implementation may involve programming the service using a standard programming language such as Java or C#
- Alternatively, services may be developed by implementing service interfaces to existing components or to legacy systems
- Services then have to be tested by creating input messages and checking that the output messages produced are as expected
- Deployment involves publicizing the service and installing it on a web server

Service implementation and deployment…

- If the service is intended to be publicly available, it has to have information for external users of the service to decide
  - if the service is likely to meet their needs and if they can trust you, as a service provider, to deliver the service reliably and securely.
- The service description should include
  - Information about *your business, contact details, etc*. This is important for trust reasons. Users of a service have to be confident that it will not behave maliciously.
  - An informal description of the *functionality provided* by the service. This helps potential users to decide if the service is what they want.
  - A description of *how to use the services*. SOAP-based

Legacy system services

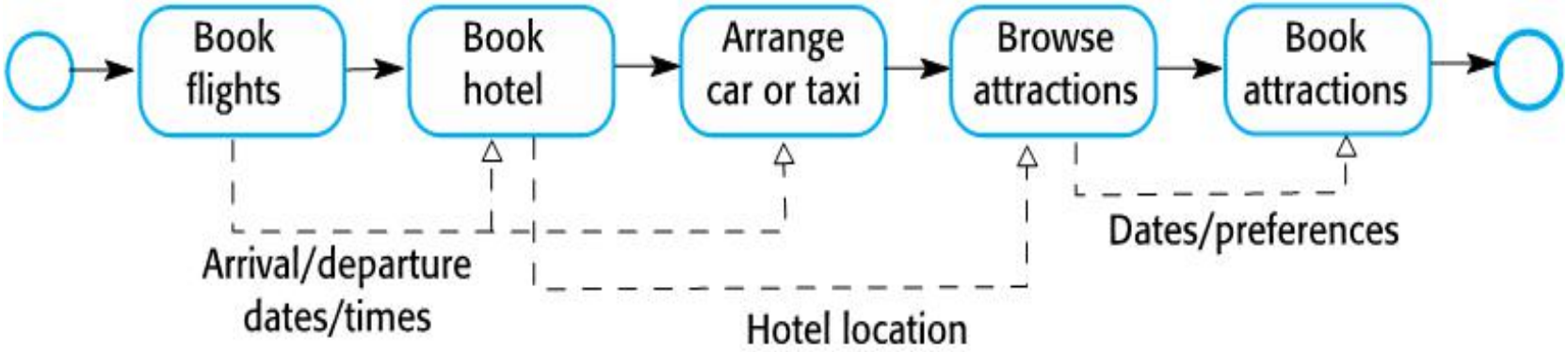
- Usually Legacy systems rely on obsolete technology but are still essential to the business.
- It may not be cost effective to rewrite or replace these systems and many organizations would like to use them in conjunction with more modern systems.
- One of the most important uses of services is to implement ‘wrappers’ for legacy systems that provide access to a system’s functions and data.
- These systems can then be accessed over the Web and integrated with other applications.

Software Development With Services

- Existing services are composed and configured to create new composite services and applications
- Service composition may be used to integrate separate business processes to provide an integrated process offering more extensive functionality.
- **Example**
  - Say an airline wishes to provide a complete vacation package for travelers.
  - As well as booking their flights, travelers can also book hotels in their preferred location, arrange car rentals or book a taxi from the airport, browse a travel guide, and make reservations to visit local attractions
  - To create this application, the airline composes its

Software Development with Services…

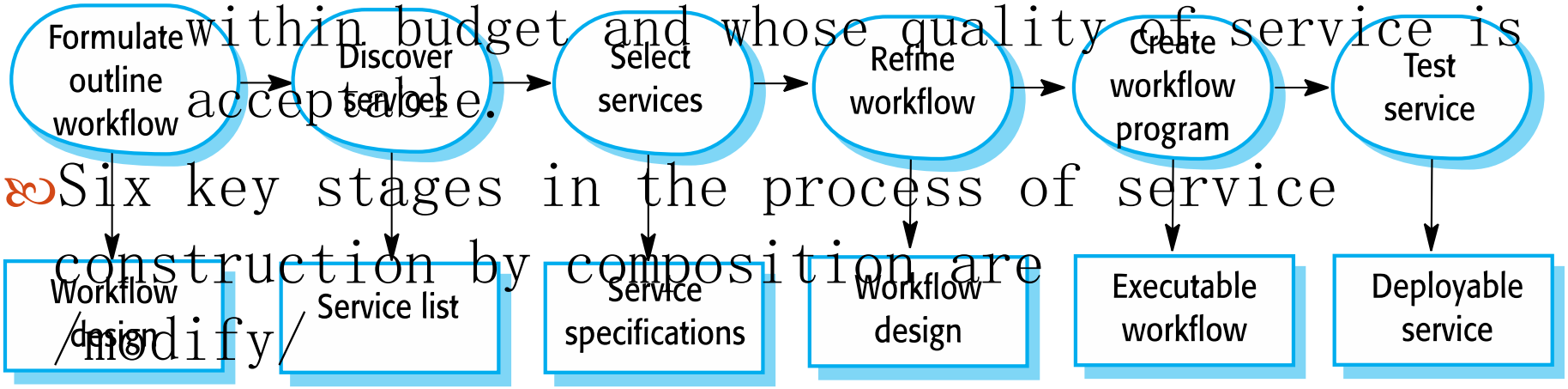
- The end result is a single service that integrates the services from different providers.



- The basis for service composition is often a workflow
  - Workflows are logical sequences of activities that, together, model a coherent business process
- Service composition is much more complex than this simple model
  - Possibility of service failure and incorporation mechanisms to handle these failures have to be considered

Software Development with Services…

- The process of designing new services by reusing existing services is essentially a process of software design with reuse
- Design with reuse inevitably involves requirements compromises
  - The 'ideal' requirements for the system have to be modified to reflect the services that are actually available, whose costs fall within budget and whose quality of service is acceptable.
- Six key stages in the process of service construction by composition are



/ modify /

Software Development with Services…

- *Formulate outline workflow*
  - In this initial stage of service design, you use the requirements for the composite service as a basis for creating an ‘ideal’ abstract service design.
- *Discover services*
  - During this stage of the process, you search service registries or catalogs to discover what services exist, who provides these services and the details of the service provision.
- *Select possible services*
  - Your selection criteria will obviously include the functionality of the services offered. They may also include the cost of the services and

Software Development with Services…

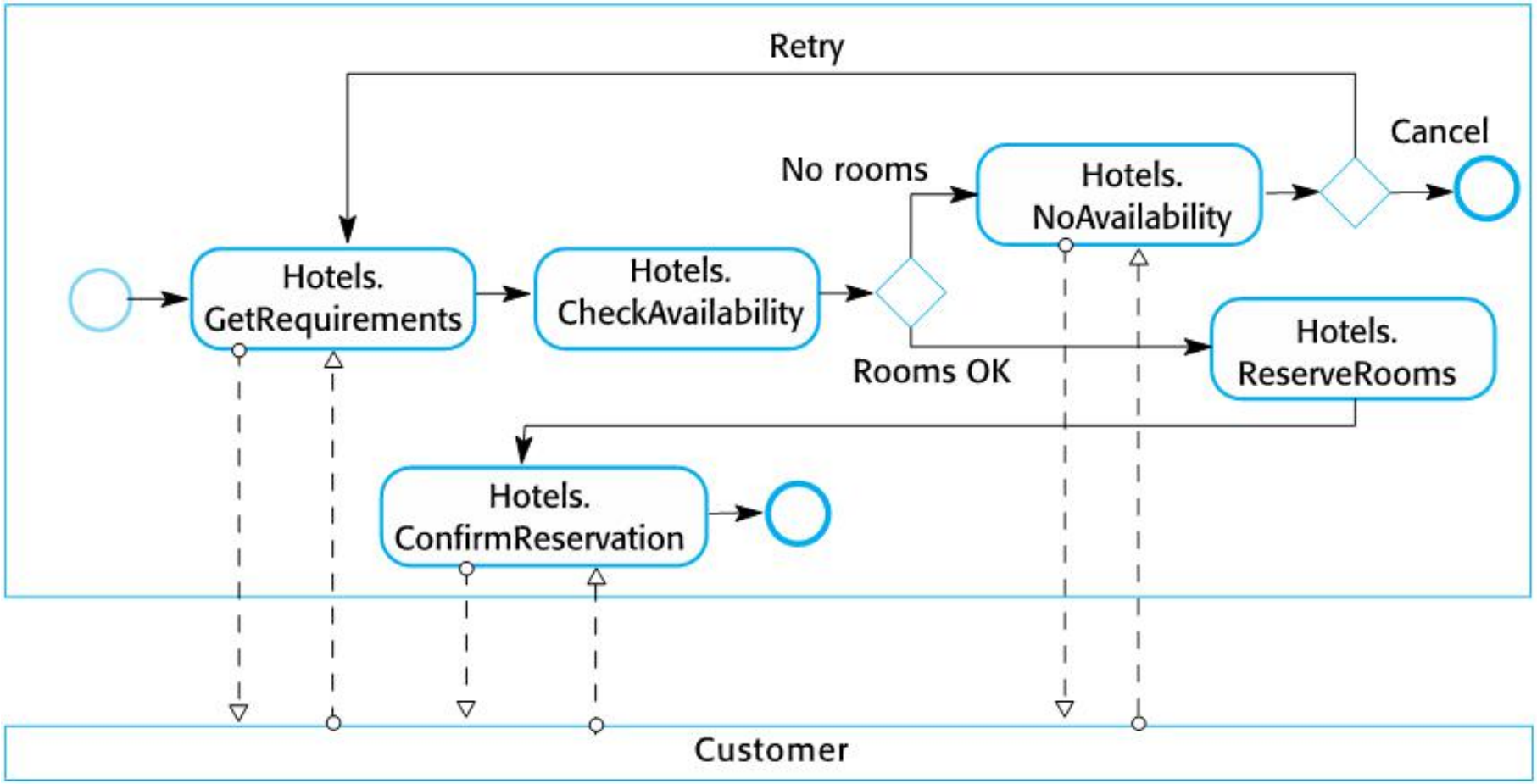
- *Refine workflow*
  - This involves adding detail to the abstract description and perhaps adding or removing workflow activities.
  - You may then repeat the service discovery and selection stages.
- *Create workflow program*
  - During this stage, the abstract workflow design is transformed to an executable program and the service interface is defined. You can use a conventional programming language, such as Java or a workflow language, such as WS-BPEL.
- *Test completed service or application*
  - The process of testing the completed, composite

Workflow design and implementation

- Workflow design involves analyzing existing or planned business processes to understand the different activities that go on and how these exchange information.
- Then the new business process in a workflow design notation is defined
- WS-BPEL is an XML-standard for workflow specification. However, WS-BPEL descriptions are long and unreadable
- Graphical workflow notations, such as BPMN, are more readable and WS-BPEL can be generated from them
- In inter-organisational systems, separate workflows are created for each organisation and

Workflow design and implementation…

- An example of a simple BPMN model of part of (hotel booking) the above vacation package

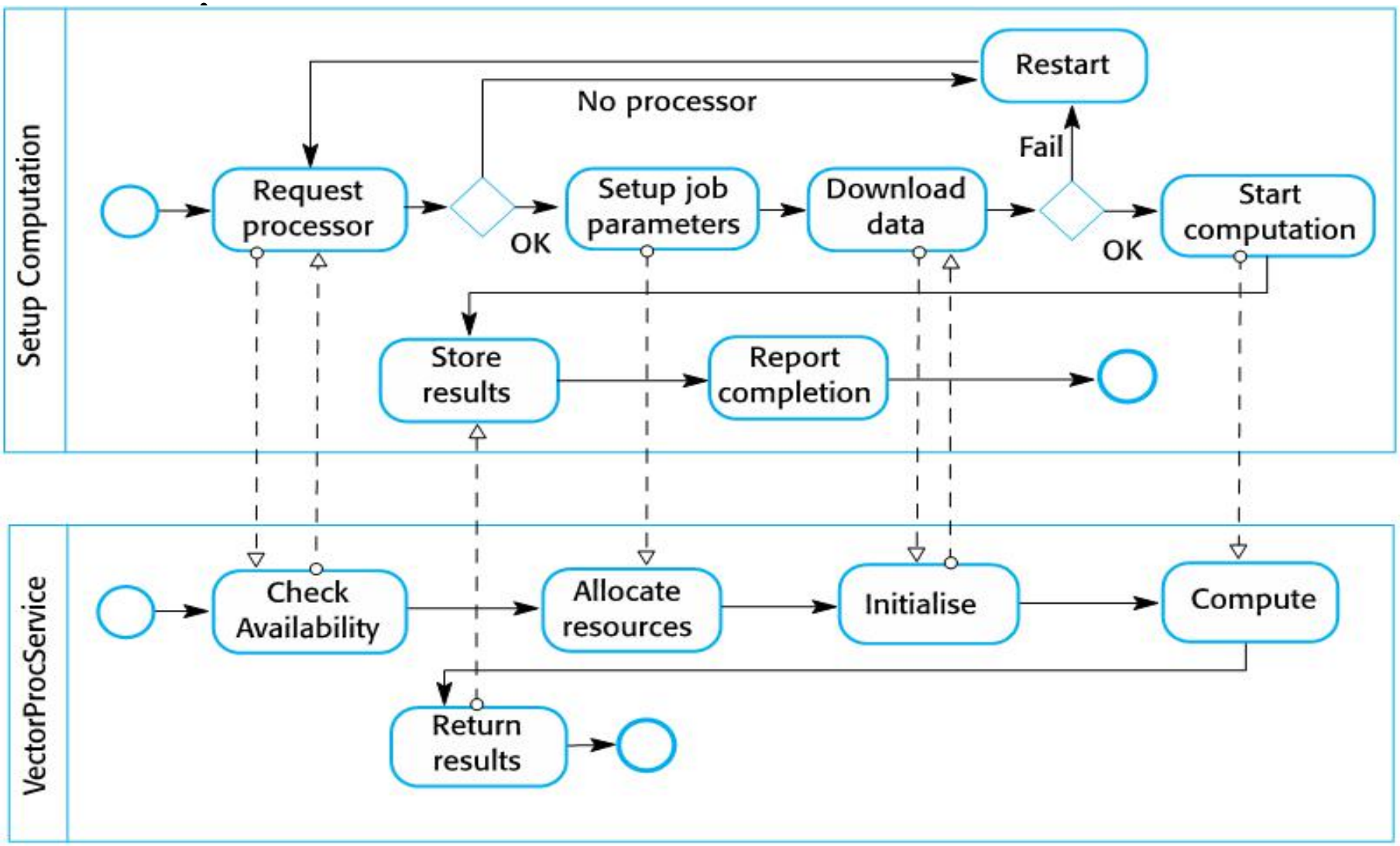


Workflow design and implementation…

- The model in preceding slide introduces some of the core concepts of BPMN that are used to create workflow models:
  - **Activities** are represented by a **rectangle** with rounded corners. An activity can be executed by a human or by an automated service.
  - **Events** are represented by **circles**. An event is something that happens during a business process. A simple circle is used to represent a starting event and a darker circle to represent an end event. A double circle (not shown) is used to represent an intermediate event.
  - A **diamond** is used to represent a **gateway**. A gateway is a stage in the process where some choice is made. For example, in the figure there is a choice made on

workflow design and implementation…

Interacting workflows - inter-organizational



Service Testing

- Testing is intended to find defects and demonstrate that a system meets its functional and non-functional requirements.
- Service testing is difficult as (external) services are 'black-boxes'
  - Testing techniques that rely on the program source code cannot be used
- **Service testing problems**
  - External services may be modified by the service provider thus invalidating tests which have been completed.
  - Dynamic binding means that the service used in an application may vary - the application tests are not, therefore, reliable.
  - The non-functional behaviour of the service is

Summary

- Service-oriented architecture is an approach to software engineering where reusable, standardized services are the basic building blocks for application systems.
- Services may be implemented within a service-oriented architecture using a set of XML-based web service standards. These include standards for service communication, interface definition and service enactment in workflows.
- Alternatively, a RESTful architecture may be used which is based on resources and standard operations on these resources.

Summary …

- Utility services provide general-purpose functionality; business services implement part of a business process; coordination services coordinate service execution.
- Service engineering involves identifying candidate services for implementation, defining service interfaces and implementing, testing and deploying services.
- The development of software using services involves composing and configuring services to create new composite services and systems.
- Graphical workflow languages, such as BPMN, may be used to describe a business process and