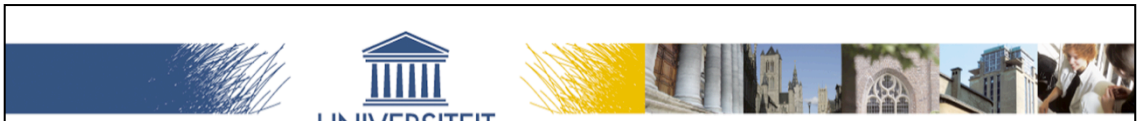
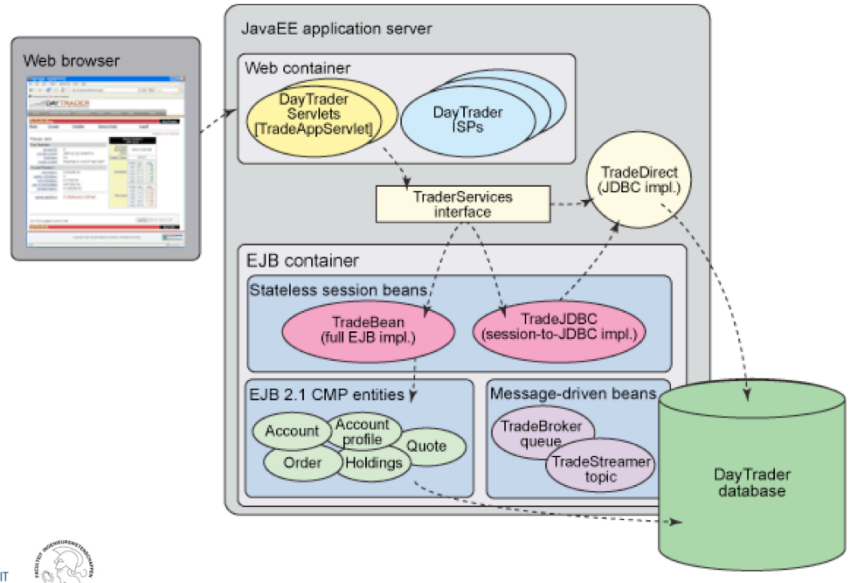


Chapter 3
Enterprise applications

Overview

1. JEE-based applications

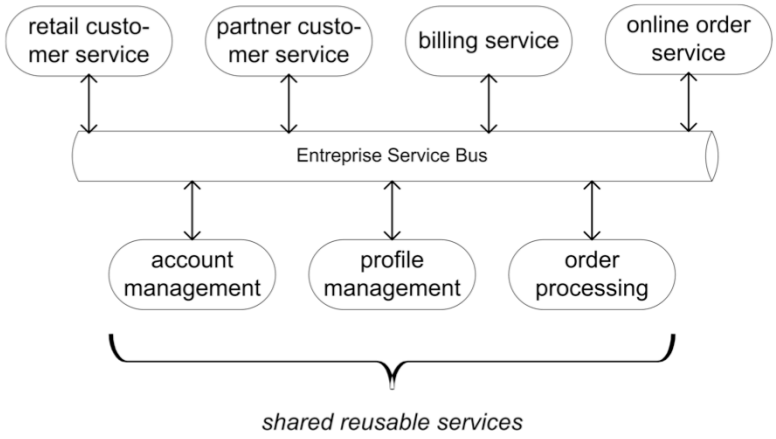


UNIVERSITEIT GENT

2

Overview

2. Service-oriented applications



for B2B applications

Outline (1)

1. Overview of Java Enterprise Edition (JEE) architecture
   - Server side architecture
   - Benefits
2. Servlets and Java Server Pages (JSP)
3. Deployment and annotations
4. Entities and Persistence
   - Managing entities
   - Primary keys
   - Object relational mappings
   - Example code
5. Enterprise Java Beans (EJBs)
   - EJB views
   - Stateless session beans
   - Stateful session beans
   - Message driven beans
6. JEE Design Patterns
   - Session Façade, Proxy and Command pattern
7. Transaction and 8. Security Service
   - Attributes and annotations

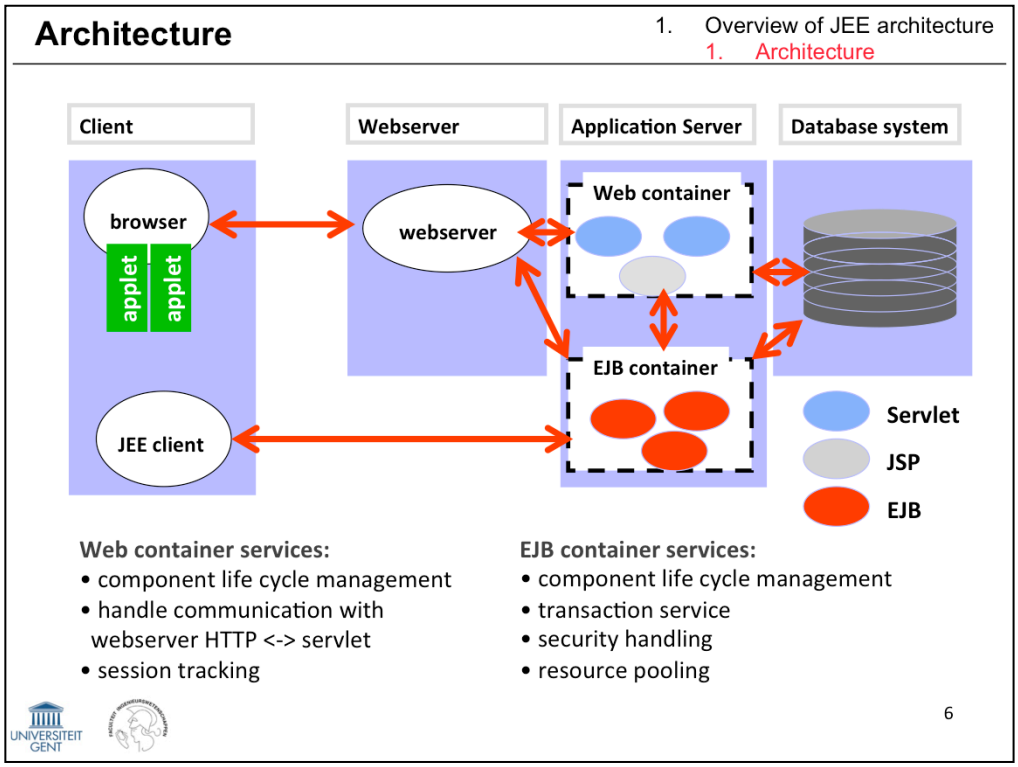
UNIVERSITEIT GENT

4

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

Outline (2)

9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12 Web Service Orchestration and Choreography
13. Software Integration
    - eTOM, ITIL

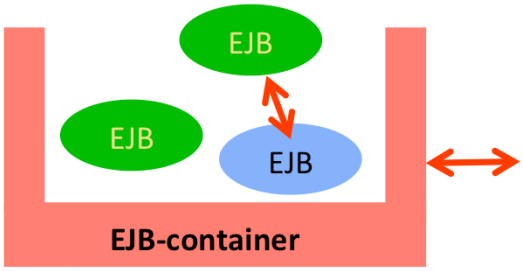


The main JEE architecture is shown in this figure. Clients are used to connect through a browser or standalone application. The webserver layer allows the clients to connect to the application (Servlet and JSP technology is used here, i.e. two technologies explained in the previous lesson). The Servlet and JSP components are run inside the webcontainer. Besides the Web container, there is also an EJB container, which runs the EJBs (Enterprise Java Beans). These are the bundled Java-components, which can be exchanged between different application servers.

The services offered by the two different containers are explained, and also detailed on the next pages.

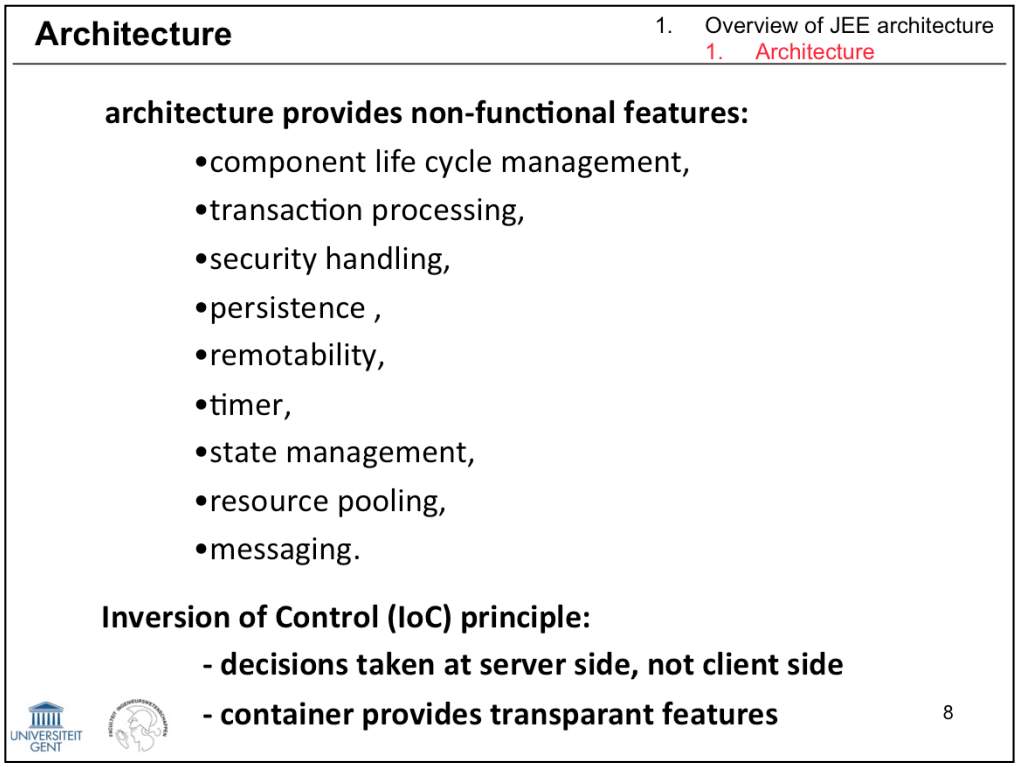
Architecture

container hosted on application server
java beans interactions mediated by container
interactions :
- with other beans
  - locally (same container)
  - remotely (different container)
- with other JEE components (servlets, jsp, ...)
- with client
- with other resources (e.g. database)



The EJB-container concept is detailed here, together with an overview of the possible interactions.

Popular EJB-containers are: JBoss, Sun GlassFish, IBM Websphere and BEA Weblogic.



The following features are provided by the EJB container (non-functional because they don't influence the functionality of an application but ease the effort for the programmer and improve the performance of the applications):

component life cycle management: starting the components when they are needed and shutting them down when no longer required

transaction processing: making sure a set of operations are executed within the context of a transaction, with no significant effort from the programmer (transactions are detailed in section 5 of this chapter)

security handling: making sure that certain method calls can only be executed when the caller of the method has the right permission (also detailed in section 5 of this chapter)

persistence: automatically storing and retrieving data from the database, without the need of the programmers to write the database query and insertion code (detailed in section 3 of this chapter)

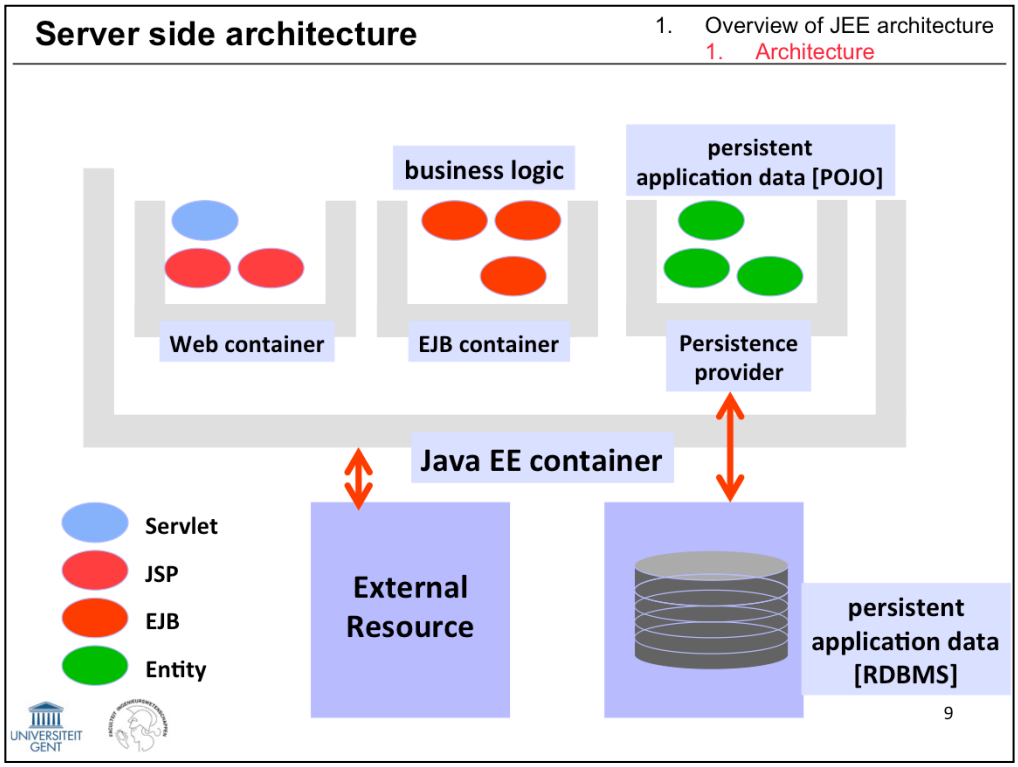
remotability: making sure components can be accessed from remote locations, without the need of writing the code to transfer method calls and data over the network

timer: allows to schedule operations

state management: maintaining the values of variables even when the components are not active

resource pooling: sharing resources over multiple requests, to make sure the resources (CPU and Memory) are used in an efficient way

messaging: providing the means to publish data and allow clients to subscribe to data they are interested in



The concept of an entity is explained here: is a Plain Old Java Object (POJO), which means it is regular Java object (not an EJB) which is run within the container of a persistence provider. A popular example of such a persistence container is Hibernate.

The remote data is stored and made accessable by a Remote Database Management System (RDBMS).

Other examples of external resources are files (local files or on a remote file system).

Benefits

1. Simplify the development of large, distributed applications:

–the EJB container provides system-level services to enterprise beans,

–the bean developer can concentrate on solving business problems.

–the EJB container--not the bean developer--is responsible for system-level services such as transaction management and security authorization.

The three main benefits of using the architecture with web containers and EJB containers are listed on this page and the next page.

Benefits (2)

2. the client developer can focus on the presentation of the client:

–because the beans--and not the clients--contain the application's business logic,

–the clients are thinner (important for clients that run on small devices)

3. enterprise beans are portable/reusable components:

–the application assembler can build new applications from existing beans.

–these applications can run on any compliant J2EE server.

Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Presentation Layer
    1. Addressing resources and HTML forms
    2. Servlets (redirection, collaboration)
    3. Session Management
    4. Java Server Pages
3. Deployment and annotations
4. Entities and Persistence
5. Enterprise Java Beans (EJBs)
6. JEE Design Patterns
7. Transaction and Security Service

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

Addressing resources : URL

<protocol>://<hostname>:<port>/<fully qualified pathname>?<parameters>

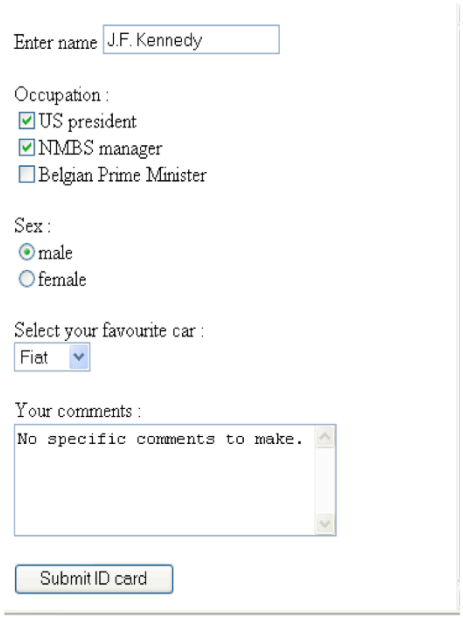
<protocol> : http, https, ftp, telnet, …
<hostname> : IP-address or DNS-name
<port> : port for webserver (default : http : 80, https: 343)
<fully qualified pathname> : directory + file name (default index.html)
<parameters> : normally used to specify (name, value)-pairs
? : separates resource name from parameters
+ : represents space
& : separate (name,value) pairs
= : binds value to name
% : escape character
followed by 2 hex-symbols representing ASCII value

elections!

e.g. **http://www.spy.cia.com:80/reports/interest?subject=elections%21**

13

HTML Web Form: example

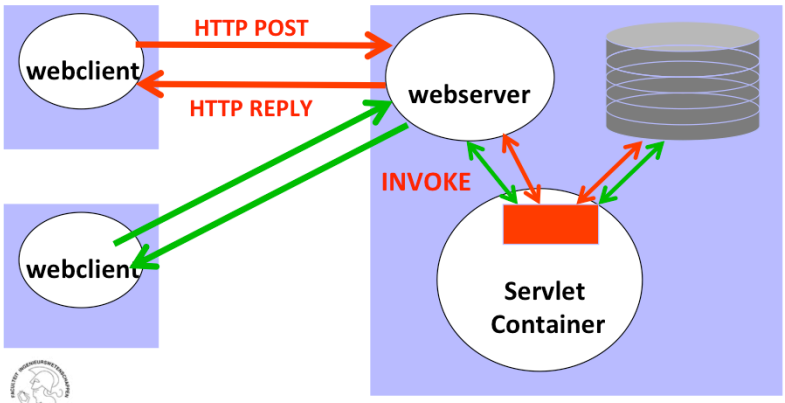


HTML Web Form: source

```
<html><head><title> Personal ID </title></head>
<body>
 <FORM METHOD="GET" ACTION="http://localhost:8080/process/input">
  <P>Enter name <INPUT TYPE="text" NAME="name" SIZE=20></INPUT></P>
  <P>Occupation :<BR>
  <INPUT TYPE="checkbox" NAME="USpres">US president<BR>
  <INPUT TYPE="checkbox" NAME="NMBS">NMBS manager<BR>
  <INPUT TYPE="checkbox" NAME="prime">Belgian Prime Minister<BR>
  </P>
  <P>Sex : <BR>
  <INPUT TYPE="radio" NAME="sex" VALUE="male" CHECKED>male<BR>
  <INPUT TYPE="radio" NAME="sex" VALUE="female">female<BR>
  <P>Select your favourite car : <BR>
  <SELECT NAME="cars">
        <OPTION VALUE="volvo">Volvo
        <OPTION VALUE="saab">Saab
        <OPTION VALUE="fiat">Fiat
        <OPTION VALUE="audi">Audi
  </select>
  </P>
  <P>Your comments :<BR>
  <TEXTAREA NAME="comments" ROWS="5" COLS="30">
  </TEXTAREA>
  </P>
  <INPUT TYPE="submit" VALUE="Submit ID card">
 </FORM>
</body>
</html>
```

Servlets and JSP

- Plug-in object at server side
- Runs separate thread per request
- Shares resources with other servlets
- Servlet life cycle managed by container process (single process, single JVM) :
  Created/destroyed/activated/passivated



Servlet life cycle

If webcontainer receives request for non-instantiated servlet :

1. Load Servlet class
2. Instantiate object of Servlet class
3. Initialize Servlet
   -> call `init()` method
4. Invoke `service()` –method to handle request
5. Handle any new requests by invoking `service()`
6. If idle-time > timeout or request by admin
   Remove servlet from webcontainer
   -> call `destroy()` method (clean up)

Servlet types

GenericServlet

```
abstract public void service(ServletRequest req,ServletResponse res)
                throws ServletException, IOException
```

- method called for each request
- method not synchronized by default !

HttpServlet

Inherits from GenericServlet
Overrides service-method

- automatically dispatches to HTTP-message related methods

```
public void doGet(HttpServetRequest req,HttpServletResponse res)
        throws ServletException, IOException
public void doPost(HttpServetRequest req,HttpServletResponse res)
        throws ServletException, IOException
```

Servlet framework methods

Automatically called methods

`public void init(ServletConfig config)`

- called by container when Servlet is loaded
- if overriden : first statement should be super.init(config)
- calls init() method

(better to override init() !)

`public void init()`

- called by init(ServletConfig)
- code non-standard initialization here (e.g. Open database connection)

`public void destroy()`

- called by container when Servlet unloaded or timeout
- locate clean-up code here

(close file handles, database connections, sockets, etc.)

- DANGER : make this method thread-safe !!!

`public abstract void service(ServletRequest req, ServletResponse res)`

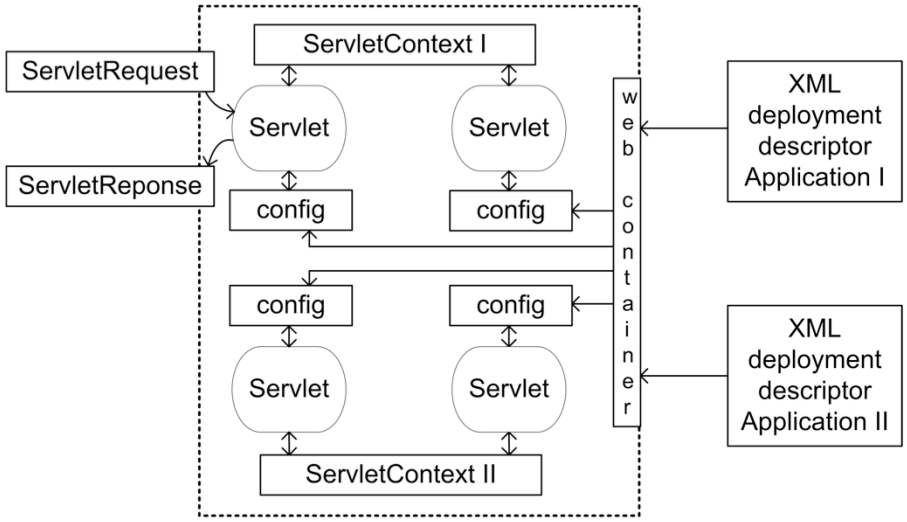
- called when request received
- overriden in HttpServlet class to dispatch to HTTP-related methods

Convenience method

`public void log(String log)`

- write (error)message to container specific log file

Servlet environment



Servlet Example

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ProcessInput extends HttpServlet  {
    public void doGet(HttpServletRequest req,HttpServletResponse res)
                                throws ServletException,IOException {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("<HTML><HEAD><TITLE>Echo back to confirm.</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("You have entered following data :<BR>");
        out.println("Your name : "+req.getParameter("name")+"<BR>");
        out.print("Your occupations : ");
        if(req.getParameter("USpres")!=null) out.print("US president ");
        if(req.getParameter("NMBS")!=null) out.print("NMBS manager ");
        if(req.getParameter("prime")!=null) out.print("Belgian Prime Minister ");
        out.println("<BR>");
        out.println("Your sex : "+req.getParameter("sex")+"<BR>");
        out.println("Your favourite car : "+req.getParameter("cars")+"<BR>");
        out.println("Your comments : "+req.getParameter("comments")+"<BR>");
        out.println("</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

Servlet Example: output

Browser output :

You have entered following data :
Your name : J.F. Kennedy
Your occupations : US president NMBS manager
Your sex : male
Your favourite car : fiat
Your comments : No specific comments to make.

Servlet redirection

```
HttpServletResponse :

public sendRedirect(String url)
              e.g. sendRedirect(http://newhost/index.html)
```

Allows distributing requests over the available servlets on different servers

Servlet collaboration

Sharing data

Within same application : use ServletContext attributes

With another application/context ON SAME SERVER :

- **use database**
- **use external object (on same or different server)**
- **use ServletContext of other application**

```
ServletContext ownContext=getServletContext();
ServletContext otherContext=
        ownContext.getContext("/otherapp/index.html");
Object parameter=otherContext.getAttribute("foreignAttribute");
```

Sharing control

Use `RequestDispatcher` to forward/include to component contents

Servlet chaining / forwarding

- make sure all headers are set appropriately before forwarding requests !

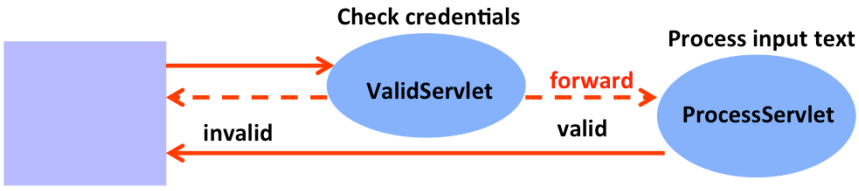
Use getRequestDispatcher()-methods

`forward(ServletRequest request, ServletResponse response)`
-> forward request to other resource, hand off control
-> response NOT committed to client !

`include(ServletRequest request, ServletResponse response)`
-> include contents to response of this servlet,
original servlet stays in control

```
// …
String destination="/servlet/NewServlet";
RequestDispatcher dispatcher=req.getRequestDispatcher(destination);
dispatcher.forward(req,res);
// …
```

Servlet forwarding : example



HTML-input form
Fields : name, password, text

Known to webserver
as "valid"

Known to webserver
as "process"

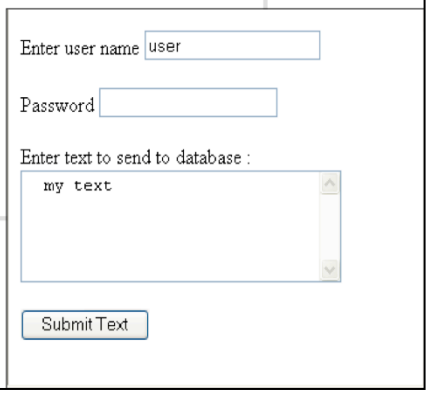
Web application "forward" contains :
- **HTML-input form "index.html"**
- **Servlet "valid" instance of class ValidServlet.class**
- **Servlet "process" instance of class ProcessServlet.class**

forward.war

Servlet forwarding : example

index.html

```
<html>
<head>
 <title> Forward </title>
</head>

<body>
 <FORM METHOD="GET" ACTION="http://localhost:8080/forward/valid">
  <P>Enter user name <INPUT TYPE="text" NAME="user" SIZE=20></
INPUT></P>
  <P>Password <INPUT TYPE="password" NAME="pass" SIZE=20></INPUT></
P>
  <P>Enter text to send to database :<BR>
  <TEXTAREA NAME="text" ROWS="5" COLS="30">
  </TEXTAREA>
  </P>
  <INPUT TYPE="submit" VALUE="Submit Text">
 </FORM>
</body>
</html>
```



Servlet forwarding : example

ValidServlet.java

```
public class ValidServlet extends HttpServlet  {
    public void doGet(HttpServletRequest req,HttpServletResponse res)throws

          ServletException,IOException {
        String user=req.getParameter("user");
        String pass=req.getParameter("pass");
        if(user.equals("user")&&pass.equals("secret")) {
          RequestDispatcher dispatcher=req.getRequestDispatcher("process");
                  dispatcher.forward(req,res);
        } else {
            res.setContentType("text/html");
            PrintWriter out=res.getWriter();
            out.println("<HTML><HEAD><TITLE>Answer from ValidServlet.</TITLE></
HEAD>");
            out.println("<BODY>");
            out.println("You failed to supply valid credentials.<BR>");
            out.println("</BODY>");
            out.println("</HTML>");
            out.close();
        }
    }
    public void doPost(HttpServletRequest req,HttpServletResponse res) throws

          ServletException,IOException {
        doGet(req,res);
    }
}
```

Servlet forwarding : example

ProcessServlet.java

```
public class ProcessServlet extends HttpServlet  {
    public void doGet(HttpServletRequest req,HttpServletResponse
res)throws
        ServletException,IOException {
         String text=req.getParameter("text");
         res.setContentType("text/html");
         PrintWriter out=res.getWriter();
         out.println("<HTML><HEAD><TITLE>Answer from
                          ProcessServlet.</TITLE></HEAD>");
         out.println("<BODY>");
         out.println("Following text has been submitted :<BR>"+text);
         out.println("</BODY>");
         out.println("</HTML>");
         out.close();
      }
    public void doPost(HttpServletRequest req,
                       HttpServletResponse res)throws

        ServletException,IOException {
         doGet(req,res);
    }
}
```

Session Management

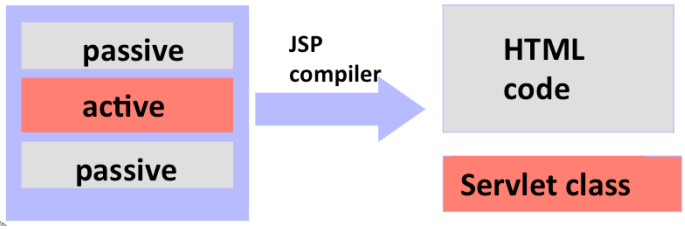
- HTTP session-less protocol

- Approaches:
  1) Cookies
  2) URL re-writing
  3) Exchange of security certificates

- Servlets:
  - automatic session management by the container
  - use `HTTPSession` interface !

Servlets and JSP

- Servlet disadvantage :
  - pages contain static + dynamic data
  - static data created dynamically -> not optimal
- Solution
  - mix static and dynamic code in single webpage
  - compile webpage to extract servlets and static portions
  - configure webserver to
    - automatically invoke servlets to create dynamic content
    - merge dynamic and static content to response to client

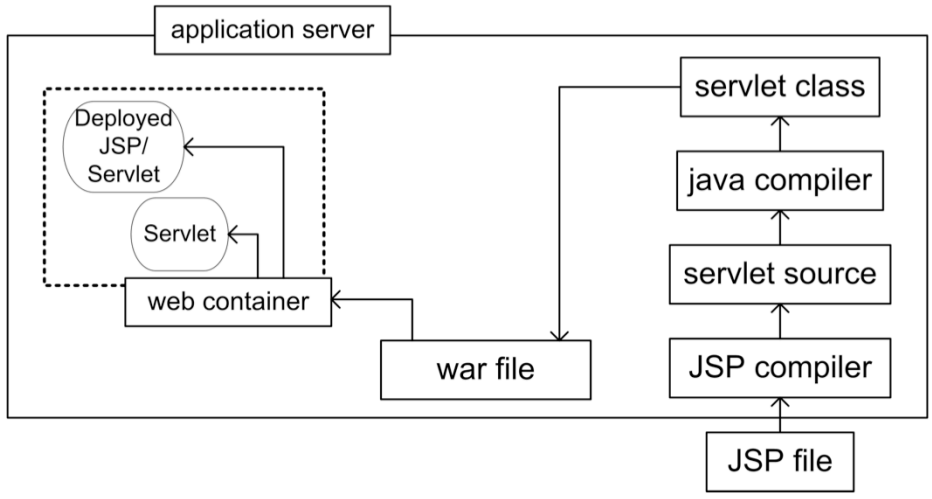


UNIVERSITEIT GENT

JSP : Java Server Page

Actually : JSP = Servlet

Java Server Pages: deployment



JSP vs Servlet: Example

Servlet version

```
import java.io.*;
import javax.servlet.*;


public class RandomGenericServlet extends GenericServlet  {
    public void service(ServletRequest req,ServletResponse res)
                throws ServletException,IOException {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        out.println("<HTML><HEAD><TITLE>Servlet Output
                 </TITLE></HEAD>");
        out.println("<BODY> Your random number is : "+
                 ((int)(Math.random()*10))+"</BODY>");
        out.println("</HTML>");
        out.close();
    }
}
```

JSP-version

```
<%@page contentType="text/html"%>
<html>
<head><title>JSP output</title></head>
<body>
Your random number is : <%=((int)(Math.random()*10))%>
</body>
</html>
```

Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Presentation Layer
3. Deployment and annotations
   - deployment
   - annotations
4. Entities and Persistence
5. Enterprise Java Beans (EJBs)
6. JEE Design Patterns
7. Transaction and Security Service

34

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

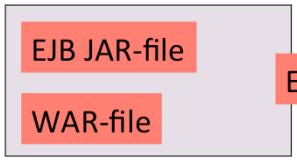
Deployment

JEE-application

enterprise bean class (bytecode) *[IDE/programmer]*
interfaces *[IDE]*
- local/remote

helper classes (utilities/exceptions/...) *[programmer]*
deployment descriptor (XML) *[IDE/deploytool]*

EJB JAR-file



EAR-file

deployable units to application server

WAR: Web Archive-file

EAR: Enterprise Archive-file

Annotations

Deployment descriptor:

- essentially contains meta-data of application/component
- cumbersome to construct/maintain
- alternative : Annotations

BUT : deployment descriptor can still override annotations

Annotations

package java.lang :

- @Deprecated
- @Override
- @SuppressWarning

own annotations can be defined

http://download.oracle.com/javase/1.5.0/docs/guide/language/annotations.html

In JEE applications, annotations are used to specify:

- mapping of attributes to database tables
- type of Enterprise Java Bean
- transaction parameters
- security parameters
- lifecycle management of entities, session beans

Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Servlets and Java Server Pages (JSP)
3. Deployment and annotations
4. Entities and Persistence – interfacing with Database Layer
   - Managing entities
   - Primary keys
   - Object relational mappings
   - Example code
   - Lifecycle management
5. Enterprise Java Beans (EJBs)
6. JEE Design Patterns
7. Transaction and Security Service

38

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

Introduction

Entity = persistent object

persistent state realised through persistent fields/properties

object/relational annotations used to map to relational data in data store

Entity class = any Java class but

annotated with `@Entity`

`public/protected` no-arg constructor

not `final` class, no persistent `final` fields, no `final` methods

if passed by value, must implement `Serializable`

persistent instance variables NOT `public`

Field is persistent, unless annotated @Transient

Primary key annotated with

- simple primary key @Id
- composite primary key @EmbeddedId, @IdClass

An definition of entity is given here. It is important to note that an entity can be any Java class (no need for instance to derive from an Entity base class or implement an entity interface) and that annotations are used to indicate the mapping of the class attributes to the tables in the data store. Annotations are indicated in source code and start with the @-sign.

From the different attributes of an entity, one needs to be selected as the primary key (i.e. a unique identifier for each particular entity instance).

Entities

- Object-oriented view of entities stored in persistent storage
  - Normally, each entity represents a row in a relational DB table
- Persistence code generated through ORM-tool (Object-Relational Mapping, e.g. Hibernate, TopLink)
- A single instance (on the server) can be ***accessed by multiple clients***
- Each instance must be uniquely identifiable by means of a *primary key*.

Primary Key

Primary Key class

must be `public`

properties `public`

public default constructor

must implements `hashCode()` and `equals()`

must be `Serializable`

composite primary key : related to

- multiple fields/properties of the entity class
- embeddable class

Same names of fields/properties as in entity class

Some specific details a primary key class has to fulfill.

Managing Entities

`EntityManager`

- find entities
- query entities
- life cycle funtions on entities (create, remove, persist)

`PersistentContext`

associated to data store

associated to its own EntityManager

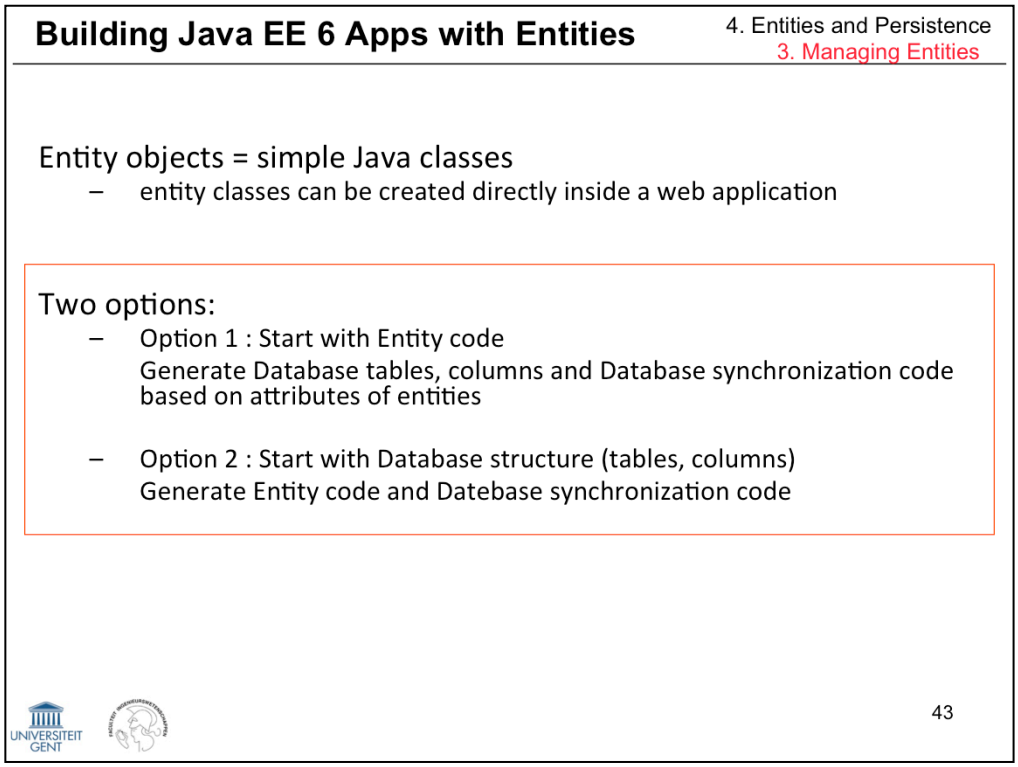
- Container Managed :
  - changes in PersistentContext automatically propagated by container
  - EntityManager obtained by @PersistenceContext annotation

    (through injection)

```
@PersistenceContext
EntityManager theManager;
```

When using entities, the EntityManager class plays an important role. The programmer can create an instance of this class to manage the available entities.

Through the @PersistenceContext annotation, an EntityManager is associated with a specific datastore (database type, host name, IP address and port).



Two options exist for building applications with entities. One is to start with the entity code, indicate the attributes to be persisted and at deployment time, the database tables are created automatically together with the database synchronization code.

The other option is to start with an existing database (structure and available data) and let an IDE generate the entity code from it. This generated code can then be adapted by the programmer when new functionality is required.

Creating the Entity Classes

Relations :

- `@OneToOne`
- `@OneToMany`
- `@ManyToOne`
- `@ManyToMany`

Every association has a direction :

- Unidirectional
- Bidirectional

Creating the Entity Classes

@OneToOne (unidirectional) 1-1

```
@Entity
public class Person implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String firstname;
   private String lastname;

   @OneToOne
   private Address address;
...
}
```

```
@Entity
public class Address implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String street;
   private String zipcode;
   private String city;
}
```

ID	Firstname	Lastname	Address_ID

ID	Street	Zipcode	City

Creating the Entity Classes

@OneToOne (bidirectional) 1-1

```
@Entity
public class Person implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String firstname;
   private String lastname;

   @OneToOne
   private Address address;
...
}
```

```
@Entity
public class Address implements Serializable {
   @OneToOne(mappedBy = "address")
   private Person person;
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String street;
   private String zipcode;
   private String city;
}
```

Owning side: Person

ID	Firstname	Lastname	Address_ID

ID	Street	Zipcode	City

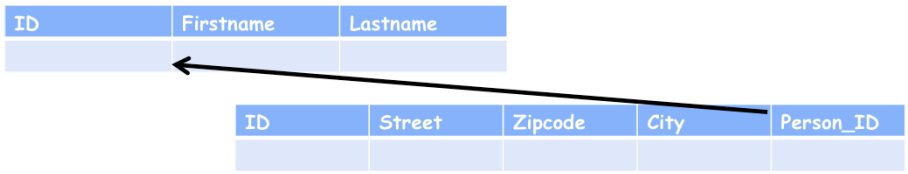
Creating the Entity Classes

@OneToOne (bidirectional) 1-1

```
@Entity
public class Person implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String firstname;
   private String lastname;

    @OneToOne(mappedBy = "person")
   private Address address;
...
}
```

```
@Entity
public class Address implements Serializable {
   @OneToOne
   private Person person;
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String street;
   private String zipcode;
   private String city;
}
```

Owning side: Address



Creating the Entity Classes

@OneToMany (unidirectional) 1-n

```
@Entity
public class Pavilion implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String name;
   private String location;

   @OneToMany
   private Collection<Animal> animals;
```

```
@Entity
public class Animal implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String name;
   private String kind;
   private String weight;
}
```

ID	Name	Location

PAVILION_ID	ANIMAL_ID

ID	Name	Kind	Weight

Creating the Entity Classes

@OneToMany (bidirectional) 1-n

```
@Entity
public class Pavilion implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String name;
   private String location;

   @OneToMany(mappedBy = "pavilion")
   private Collection<Animal> animals;
```

@ManyToOne n-1

```
@Entity
public class Animal implements Serializable {
   @ManyToOne
   private Pavilion pavilion;
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String name;
   private String kind;
   private String weight;
}
```

ID	Name	Location

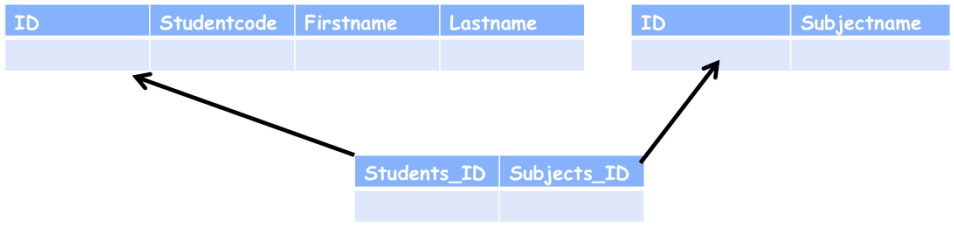
ID	Name	Kind	Weight	PAVILION_ID

Creating the Entity Classes

@ManyToMany (bidirectional) n-m

```
@Entity
public class Student implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String firstname;
   private String lastname;
   private int studentcode;

    @ManyToMany
   private Collection<Subject> subjects;
```

```
@Entity
public class Subject implements Serializable {
   @ManyToMany(mappedBy = "subjects")
   private List<Student> students;
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   private String subjectname;
```



Entity Lifecycle Management

`@PrePersist`
Marks a method in the entity class as a pre-persist callback (executed before an entity is persisted).

`@PostPersist`
Marks a method in the entity class as a postpersist callback (executed after an entity is persisted).

`@PreUpdate`
Marks a method in the entity class as a preupdate callback (executed before entity data is updated in the database).

`@PostUpdate`
Marks a method in the entity class as a postupdate callback (executed after entity data is updated in the database).

`@PreRemove`
Marks a method in the entity class as a preremove callback (executed before an entity is removed from the database).

`@PostRemove`
Marks a method in the entity class as a postremove callback (executed after an entity is removed from the database).

`@PostLoad`
Marks a method in the entity class as a postload callback (executed before an entity is loaded from the database).

Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Servlets and Java Server Pages (JSP)
3. Deployment and annotations
4. Entities and Persistence
5. Enterprise Java Beans (EJBs) – Business Layer
   - EJB views
   - EJB types
   - Stateless session beans
   - Stateful session beans
   - Message driven beans
6. JEE Design Patterns
7. Transaction and Security Service

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

Types of EJBs: overview

EJBs

Session Beans

- session related object
- always associated to one single client at most
- types
  - **Stateful** : "conversational state"
  - **Stateless**
  - **Singleton** instantiated once/app

synchronous

Message Beans

- asynchronous message handling
- new since J2EE 1.3

asynchronous

Entities

- "real life" object
- mostly associated to "row in database"
- persistent
- NEW since EJB3.0 **[replace (very) complex EntityBeans]**
- NOT managed by EJB container

Session EJBs

A session bean instance is:

- A ***non-persistent*** object
- Implements some business logic ("procedural component")
- Runs on the server

Not shared among multiple clients

Since this communication via a network, finding the server means determining the servers network address.

Stateless Session EJBs

Conceptually, the same as stateful session EJBs

No state

- ***http-style request – reply interaction***
- *Can* have fields, but they are not unique to any client

Basically, it's an ***optimization trick***:

Since the container knows the bean has no state, it can:

- Use a single bean instance (While each client thinks it has its own copy)
- Destroy/re-instantiate on the fly
- Redirect requests to different instances (load balancing)

Example: CurrencyConversionBean

Stateful Session EJBs

A stateful session bean ***maintains a state***

- values of its instance variables
- also called : conversational state

The state is ***relevant only for a single client***

- cannot be seen by other clients

The state is ***not persistent***

- does not survive a server shutdown
- when the client removes the bean or terminates, the session ends and the state disappears

Canonical example: ShoppingCart

Singleton Session EJBs

Instantiated once per application
and exists for the lifecycle of the application.

Use: single enterprise bean instance shared across and concurrently accessed by clients

Message Driven Beans (MDBs)

process messages asynchronously

messages originate from JMS (Java Messaging Service)-compliant system

message can be sent by any source

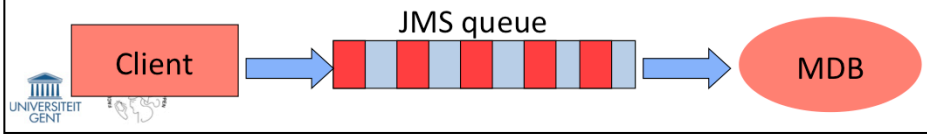
- other EJB
- other JEE component (e.g. web component)
- any other (legacy) component

indirectly accessed by clients

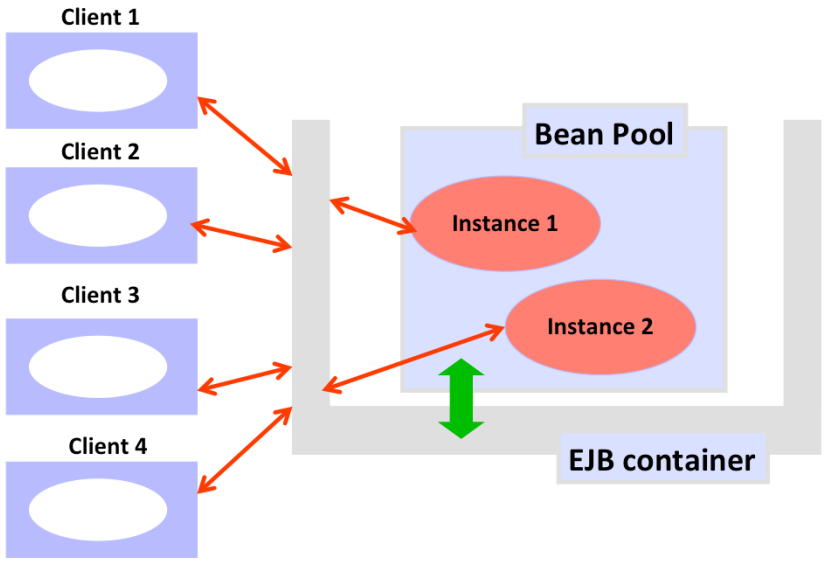
- no interface, use message instead

similar to stateless session bean

- all beans are equivalent (pooling !) – no client specific state
- no conversational state maintained
- can handle requests from multiple clients



Stateless Session Bean usage



JEE standard example Currency Converter

CurrencyConverterBean.java

```
package converter;

import java.math.BigDecimal;
import javax.ejb.Stateless;

@Stateless
public class CurrencyConverterBean implements
   CurrencyConverterRemote {
    BigDecimal yenRate = new BigDecimal("121.6000");
    BigDecimal euroRate = new BigDecimal("0.0077");

    public CurrencyConverterBean() {
    }

    public BigDecimal dollarToYen(BigDecimal dollars) {
        BigDecimal result = dollars.multiply(yenRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }

    public BigDecimal yenToEuro(BigDecimal yens) {
        BigDecimal result = yens.multiply(euroRate);
        return result.setScale(2, BigDecimal.ROUND_UP);
    }

}
```

Currency Converter : remote interface

```
package converter;

import java.math.BigDecimal;
import javax.ejb.Remote;


@Remote
public interface CurrencyConverterRemote {
    BigDecimal dollarToYen(BigDecimal dollars);

    BigDecimal yenToEuro(BigDecimal yens);

}
```

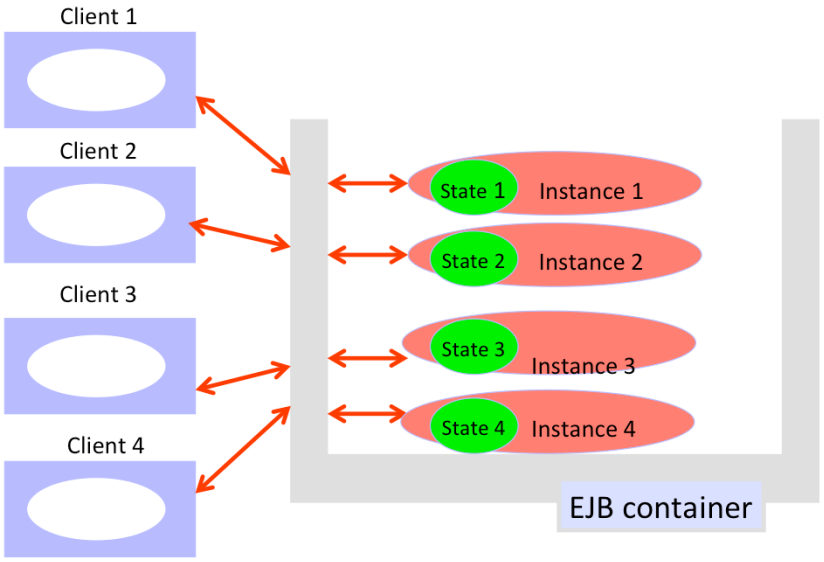
CurrencyConverterRemote.java

Application Client

```
package currencyconverterclient;

import converter.CurrencyConverterRemote;
import java.math.BigDecimal;
import javax.ejb.EJB;
public class Main {
    @EJB
    private static CurrencyConverterRemote currencyConverterBean;

    public Main() {
    }

    public static void main(String[] args) {
            BigDecimal param = new BigDecimal("100.00");
            BigDecimal amount =
               currencyConverterBean.dollarToYen(param);
             System.out.println(amount);
            amount = currencyConverterBean.yenToEuro(param);
            System.out.println(amount);
    }

}
```

Main.java

Stateful Session Bean usage



typical usage : multi-step workflows

Activation and Passivation

During the bean's lifecycle, the container may decide to *passivate* it

Normally done to free up memory and other resources

Bean is notified immediately before passivation by lifecycle method `@PrePassivate`

In this method, the bean should:

- Release any heavy-duty resources it might be holding
  - Open files, database connections, etc.
- Nullify fields that should not be serialized (cache, etc.)
- Copy state into `Serializable` variables

If the bean is referenced by client while passivated, the container *activates* it.

Bean is notified by `@PostActivate` immediately after de-serialization

- Its chance to restore all field values, resources, etc.

Stateful Session Beans: Programming Rules

Additional Rules:

- `@Stateful` (instead of `@Stateless`)
  identical (except for name) as `@Stateless`
- instance variables must be primitive/`Serializable`
- `@Remove` for "friendliness"
- provide additional methods (wrt stateless beans) :
  - `@PostActivate`
  - `@PrePassivate`

Simple Example : a random bean

- business methods:
  - **random()** : give a random int number between 0 and 10
  - **random(int n)** : idem between 0 and n
  - **getSum()** : get sum of all values produced *in this session*
  - **getUser()** : get user of *this session*
  - **setUser()** : set user of *this session*

Random State Bean : remote interface

RandomStateRemote.java

```
package randomstate;

import javax.ejb.Remote;
import javax.ejb.Remove;

@Remote
public interface RandomStateRemote {
    int random(int n);
    int random();
    void setUser(String name);
    String getUser();
    int getSum();
    @Remove
    void remove();
}
```

Random State Bean : Bean Class

RandomStateBean.java

```
package randomstate;
import javax.ejb.Remove;
import javax.ejb.Stateful;

@Stateful
public class RandomStateBean implements RandomStateRemote
{
    private String userName;
    private int sum=0;
    public RandomStateBean() {}
    public int random(int n) {
        int r=(int)(n*Math.random());
        sum+=r;
        return r;
    }
    public int random() {return random(10);}
    public void setUser(String name) {userName=name;}
    public String getUser() {return userName;}
    public int getSum() {return sum;}
    @Remove
    public void remove() {userName=null;sum=0;}
}
```

session related state variables

friendly removal

Random State Bean : multisession client

Main.java

```
package twosessionclient;
import javax.ejb.EJB;
import randomstate.RandomStateRemote;
public class Main {
    @EJB
    private static RandomStateRemote george;
    @EJB
    private static RandomStateRemote john;
    public Main() {}

    public static void main(String[] args) {
        george.setUser("George");
        john.setUser("John");
        for(int i=0;i<5;i++)
            System.out.println("1:"+george.random(10)+
                                    "\t2:"+john.random(100));
        System.out.println(george.getUser()+" : "+george.getSum());
        System.out.println(john.getUser()+" : "+john.getSum());
        george.remove();
        john.remove();
    }
```

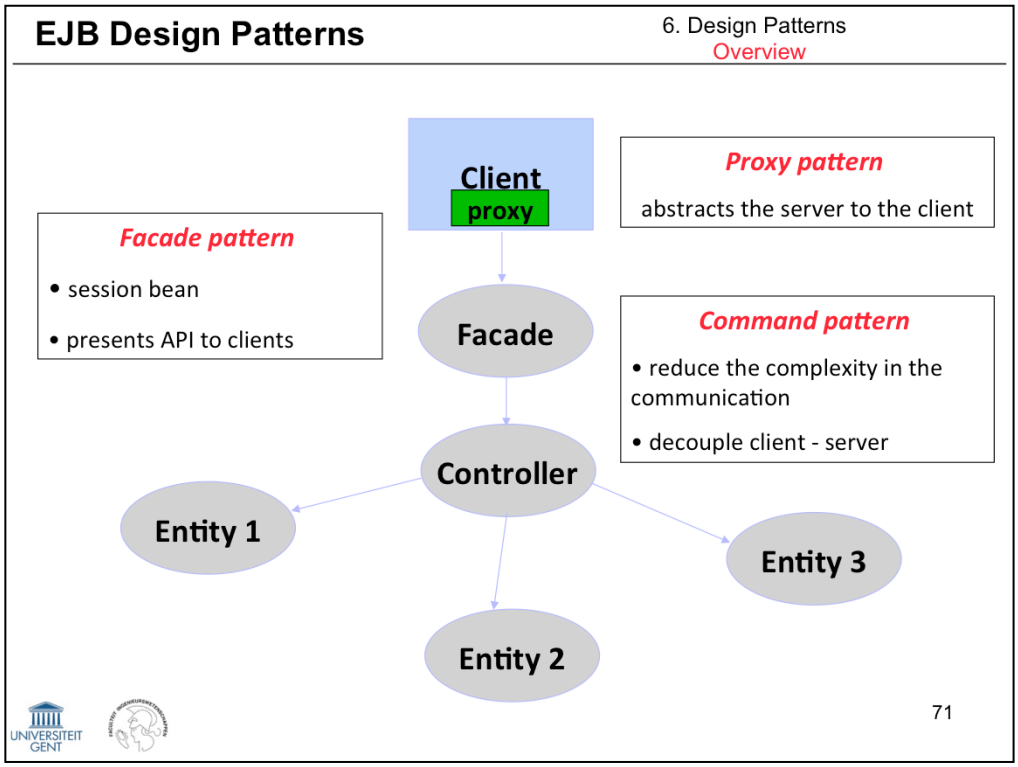
sample run

```
1:5    2:8
1:0    2:25
1:6    2:43
1:3    2:34
1:1    2:1
George:15
John:111
```

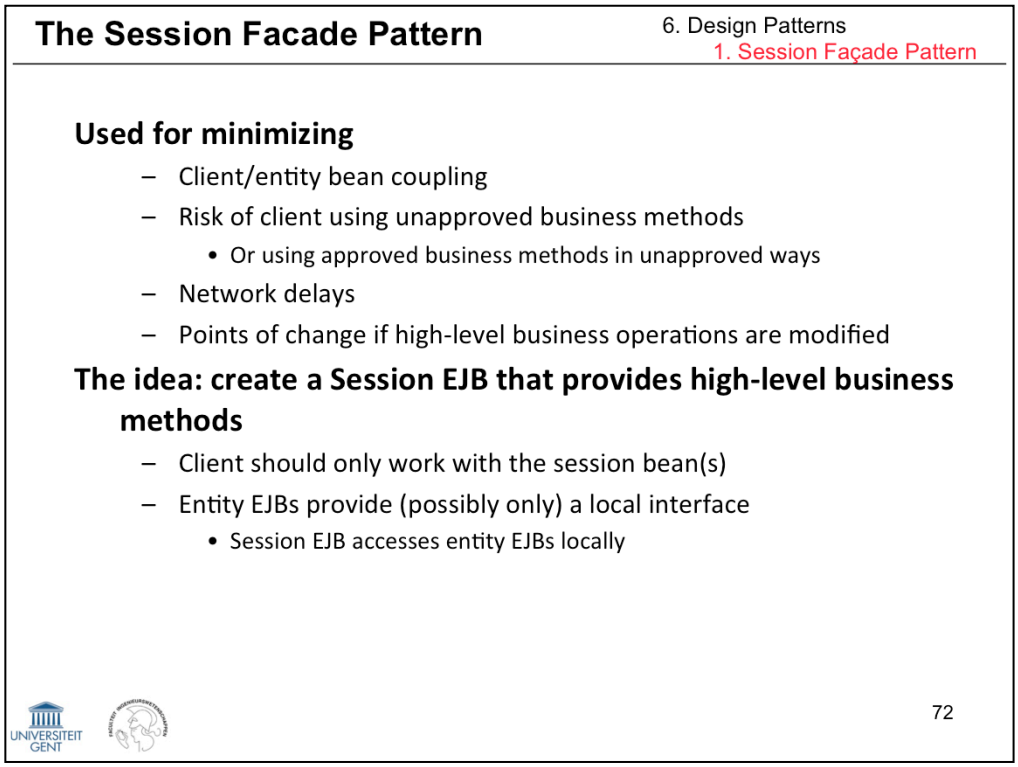
Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Servlets and Java Server Pages (JSP)
3. Deployment and annotations
4. Entities and Persistence
5. Enterprise Java Beans (EJBs)
6. JEE Design Patterns
   - Session Façade, Proxy and Command pattern
7. Transaction
8. Security Service

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.



Three JEE-specific design patterns are detailed in this section: the façade pattern, proxy pattern and command pattern.



The definition and advantages of the façade pattern are described.

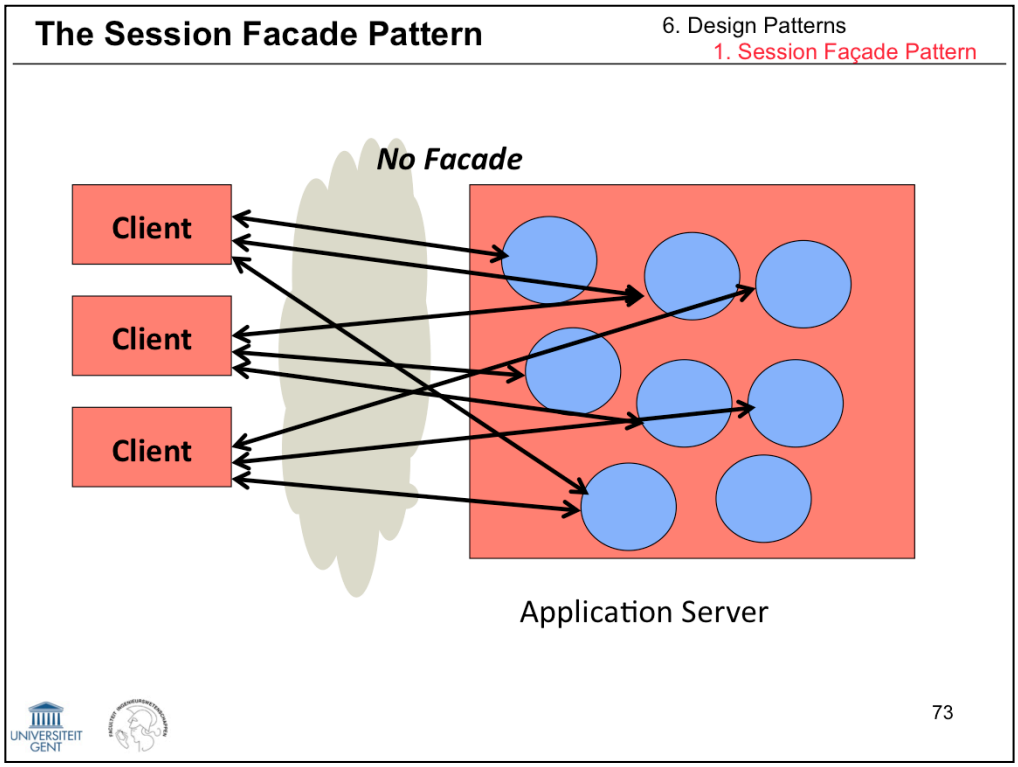
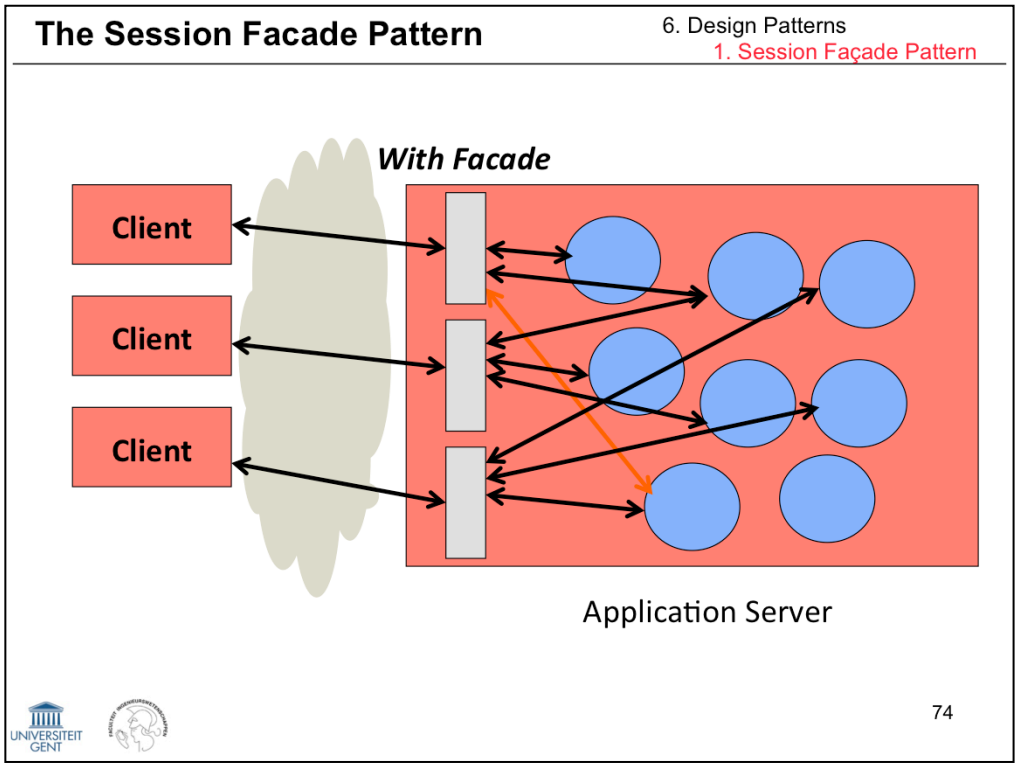


Illustration of the communication between the clients and entities when no façade session bean is used.



Similar illustration of the communication between the clients and entities, in this case a façade session bean is used.

Session Facade Example

Client code without the facade:

- Find `Account` home
- Find accounts `a1`, `a2` by primary keys `k1`, `k2`
- `a1.setBalance(a1.getBalance() - amount);`
- `a2.setBalance(a2.getBalance() + amount);`

Client code with the facade:

- Find `TransferBean` home, create `TransferBean t1`
- `t1.transfer(k1, k2, amount);`

Illustration of the usefulness of a façade session bean for a specific example.

Resulting Benefits

Faster operation

- 1 JNDI lookup, 1 remote object lookup and 1 remote business method *vs.* 1 JNDI lookup, 2 remote object lookups and 4 remote business methods in the old code

Easier maintenance

- If we have two different clients (servlet, application) and we wish to change the transfer semantics (e.g., add logging), there's only one place to change

Easier transaction management

- can be started at the facade

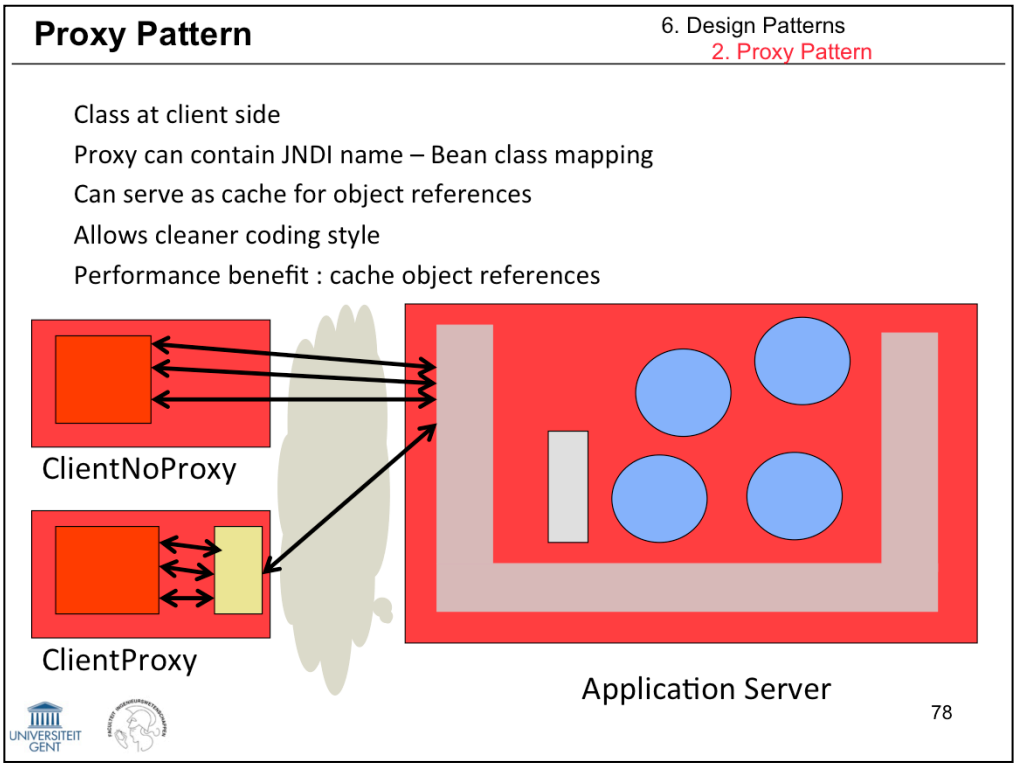
Increased security

- Client can't invoke `Account.setBalance` directly, since `Account` has no remote component interface

List of benefits when using the façade session bean design pattern.

Example Façade Session


```
package ods.ejb;

import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.NoResultException;
import javax.persistence.PersistenceContext;

@Stateless
public class AccountFacade implements AccountFacadeLocal {

    @PersistenceContext
    private EntityManager em;

public AccountFacade() {
    }

public boolean isValidAccount(String username, String
     password){
        Account account = findByUsername(username);
        boolean valid = false;

        If(account != null)
            if(account.getPassword().equals(password))
                valid = true;
        return valid;
    }

public Account addAccount(String username,
                      String password, boolean isAdmin)
    throws IllegalArgumentException{

      Account existing = findByUsername(username);
        if(existing != null)
            throw new IllegalArgumentException("username
     allready exists");
        else{
            Account account = new Account(username,
     password, isAdmin);
            create(account);
            return account;
        }
    }

 public Account findByUsername(String username){
        try {
            Account account = (Account)
    em.createNamedQuery("findAccountByUsername")
            .setParameter("username",
    username).getSingleResult();
            return account;
        } catch (NoResultException e) { }
        return null;
    }

public void create(Account account) {
        em.persist(account);
    }

public void edit(Account account) {
        em.merge(account);
    }

public void destroy(Account account) {
        em.remove(em.merge(account));

}

public Account find(Object pk) {
        return (Account) em.find(Account.class, pk);
    }

public List findAll() {
        return em.createQuery("select object(o) from
     Account as o").getResultList();
    }

}
```



Explanatory figure to illustrate the Proxy design pattern for JEE applications.

Command Pattern

Command pattern

- = Class, transmitted from one object to another
- Client constructs commands, facade forwards them to right controller

Example code

```
public interface BasicCommandInterface{
    public String getCommandTarget();
    public String getCommandInstruction();
    public HashMap getCommandData();
    public void setCommandData(String key,Object value);
    public void setCommandTarget(String aTarget);
    public void setCommandInstruction(String anInstruction);
    public execute();
    public void getCommandData(HashMap aData);
}
```

Applications can be modified dynamically

Facade doesn't need to provide all interface methods

UN

When the command pattern is used, an object is transmitted between the client and session bean. This object contains the command information in a textual format. This approach is more flexible than the approach that for each possible command a separate method is available (since adding or changing a command requires to change the session bean interface and implementation).

In the case of the command design pattern, the client constructs the commands and the facade forwards them to right controller, where the commands are interpreted and forwarded.

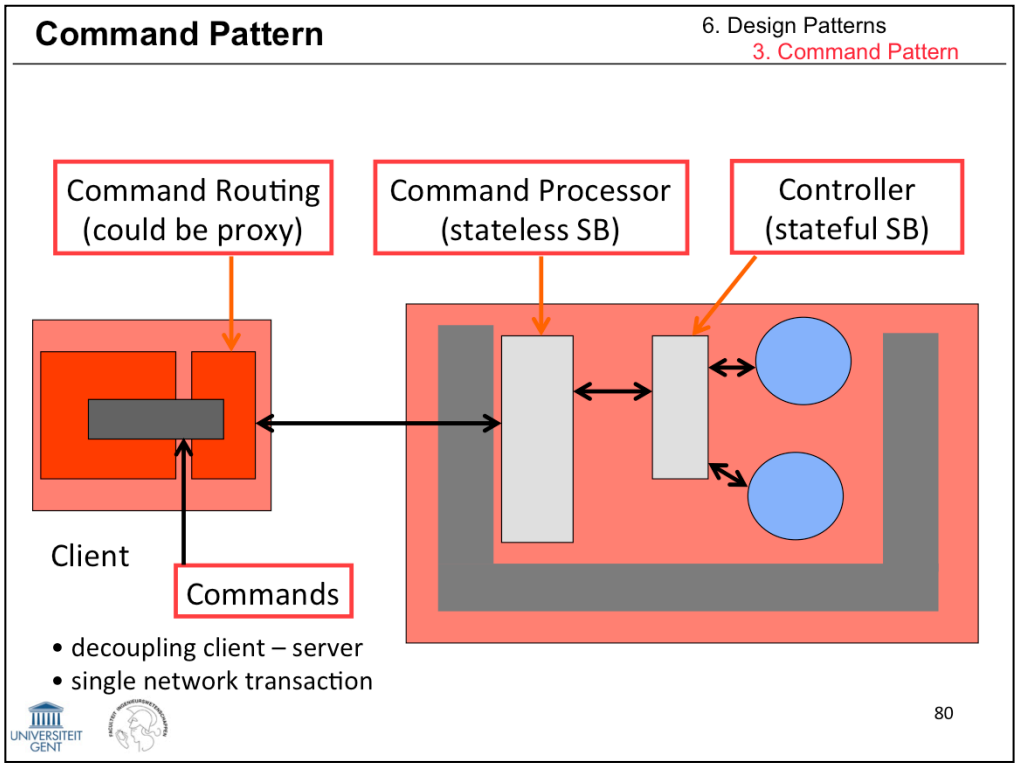


Illustration of the communication between the clients, the session beans and the entities when the command pattern is used.

Other Recommended Practices

Avoid calling EJB methods directly from JSP

- The need to catch remote exceptions would complicate the JSP code

Avoid hard-coding JNDI resource names

Avoid fine grain data transfers

To conclude this section, some extra recommended practices are listed.

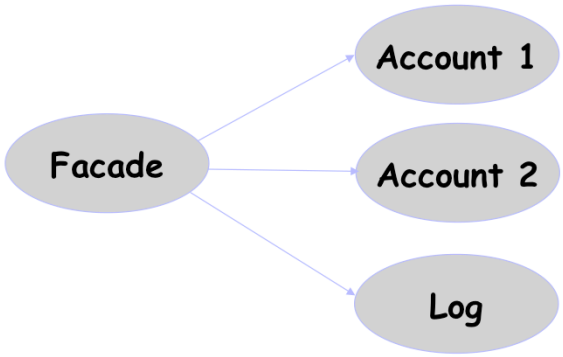
Outline

1. Overview of Java Enterprise Edition (JEE) architecture
2. Servlets and Java Server Pages (JSP)
3. Deployment and annotations
4. Entities and Persistence
5. Enterprise Java Beans (EJBs)
6. JEE Design Patterns
7. Transaction and 8. Security Service
   - Introduction
   - Transaction attributes
   - Transaction rollback
   - Transaction annotations
   - Security terminology
   - Declarative security
   - Programmatic security

82

After an overview of the basic concepts (programming language generations and compilation versus interpretation), the following programming paradigms are detailed: procedural, functional, object oriented and declarative. Moreover, scripting languages will be explained. The next three sections focus on an overview of Java, C/C++ and C# and their important concepts.

Introduction



Set of operations need to move data from one consistent state to another (example shown: transfer money from one account to another and log that the transfer is done)

The set of operations should be indivisible

If one or more operations fail, the entire set should be undone

The

Possible Approach

If one or more operations fail, the entire set should be undone:

By means of exception handling:
- Catch all possible exceptions of involved methods
- In case of exceptions: undo already finished operation

Drawbacks:
- Code spread across many exception handlers
  - Less readable
  - Less maintainable
- In case of network failures
  - Has the operation been completed or not ?
  - Remote object should roll back automatically if the network becomes unavailable

=> Use of transactions

What is a Transaction?

A transaction is *a set of operations that moves data from one consistent state to another*

If one or more operations fail, the entire set is undone

- Success: the transaction "commits"
- Failure: the transaction "rolls back"

For example:

```
begin transaction
   debit checking account
   credit savings account
   update history log
commit transaction
```

Transaction Definitions

Transactional Object: An object which is used (methods invoked) within a transaction

- Can only be associated with one transaction at a time
- EJBs can be transactional objects

Transactional Client: A program which invokes methods on transactional objects

Transaction Manager: A program that coordinates transaction processing

The Transaction Context

A single transaction can involve multiple objects and operations

A *transaction context* represents the transaction shared by all these participants

By default, it is automatically propagated between transactional objects (EJBs).

Java APIs for Transactions

The *Java Transaction API* (JTA) is used by application developers

- Specifies the interface between the transaction manager and all involved objects
- Main class: the `UserTransaction` interface.

The *Java Transaction Service* (JTS) is used by developers of transaction managers

- Developers of application servers, EJB containers, etc.
- Not used by application developers

Container Managed Transactions

Can be used with both session beans and entities.

The container begins a transaction immediately before an enterprise bean method starts (if declared to be transactional).

The transaction is committed before the method exits

Not all methods should be associated with transactions

=> Use of transaction attributes at deployment time

Transaction Attribute Values

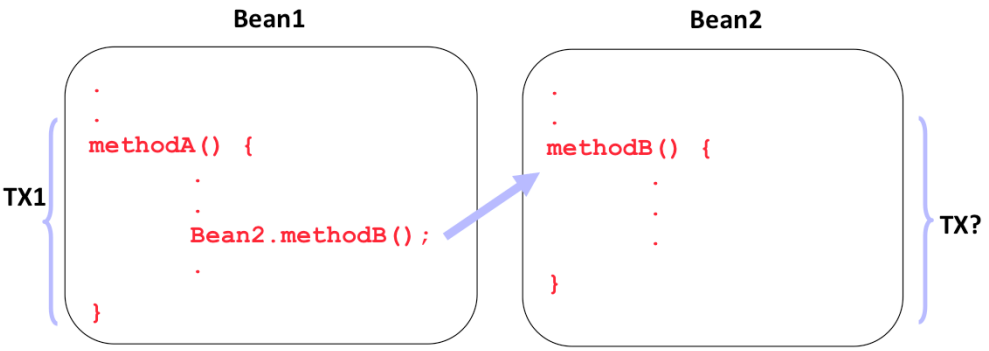
When using CMT, beans (or specific methods within beans) have a transaction attribute

Attribute controls ***scope*** of transaction

One of six possible values:

- `Required`
- `RequiresNew`
- `Mandatory`
- `Never`
- `NotSupported`
- `Supports`

Transaction Attributes



methodB() part of TX1 ?
(dependent on transaction attribute, associated with methodB detailed on next 6 slides !)

Required

`Required`

- If the invoker has a transaction context, it is propagated to the bean
- If not, the container creates a new transaction context
- I.e., the method always executes within a transaction context, but does not create a new transaction needlessly
- *This is the default value*

	Bean1 methodA()	Bean2 methodB()
Transactional Context	TX1 None	TX1 TX2

RequiresNew

RequiresNew

- If the invoker has a transaction context, it is suspended for the duration of this method's execution
- Either way, a new transaction context is created
- Use when you want the method to execute within a transaction, but you do not want failure to roll back the whole transaction

	Bean1 methodA()	Bean2 methodB()
Transactional	TX1	TX2
Context	None	TX2

Mandatory

`Mandatory`

- If the invoker has a transaction context, it is propagated to the bean
- Otherwise, an exception is thrown
  - `TransactionRequiredException`
- Use when you want the invoker to provide the transaction
  - Does not necessarily imply BMT or client managed transactions: the invoker can be a different method in the same EJB

	Bean1 methodA()	Bean2 methodB()
Transactional	TX1	TX1
Context	None	**EXCEPTION**

Never

`Never`

- If the invoker has a transaction context, an exception is thrown
  - **`RemoteException`** `or` **`EJBException`**
- Otherwise, the method proceeds normally, without a context
- Used for non-transactional resources

	Bean1 methodA()	Bean2 methodB()
Transactional	TX1	**EXCEPTION**
Context	None	**None**

NotSupported

`NotSupported`

- If the invoker has a transaction context, it is *suspended* for the duration of this method's execution
- If not, no transaction context is used
- Either way, the method executes without transaction context
- For performance optimization

	Bean1 methodA()	Bean2 methodB()
Transactional	TX1	**TX1 suspended**
Context	None	**None**

Supports

`Supports`

- If the invoker (client, or a different method) has a transaction context, it is propagated to the bean
- If not, no transaction context is used
- Use this for "don't care" situations

	Bean1	methodA()	Bean2 methodB()
Transactional	TX1		TX1
Context	None		**None**

Transaction Attribute Values: Summary

	Invoker's transaction context	Method's action
Supports	T_1	Use T_1
	None	Work with no transaction context
NotSupported	T_1	T_1 suspended
	None	Work with no transaction context
Required	T_1	Use T_1
	None	Create and use T_2
RequiresNew	T_1	T_1 suspended, create and use T_2
	None	Create and use T_2
Mandatory	T_1	Use T_1
	None	Throw exception
Never	T_1	Throw exception
	None	Work with no transaction context

Recommended Practices

Always prefer container managed transactions.

Prefer `Required`, `RequiresNew` and `Mandatory` over other options

- `Supports`, `NotSupported` and `Never` are not implemented by all containers - compromises portability

Exceptions and Transactions

An application exception does not abort the transaction

- However, note that if the exception causes the method that initiated the transaction to end, then the transaction *will* be stopped
- If the client invokes m_1, m_1 initiates the transaction (due to its Requires attribute) and invokes m_2, and m_2 throws an application exception, then m_1 can try to continue the transaction
- If m_1 throws (or does not catch) an application exception, the transaction will roll back

A system exception *aborts the transaction*

- The container will log the error and mark the transaction "rollback only"
- Any attempt to proceed with the transaction would be futile

Rolling back a Transaction

1. By the container: if a system exception is thrown

2. The bean method can instruct the container to roll back a transaction

   By invoking `setRollBackOnly` on its EJB context object

   Usually done before throwing an application exception

Roll back example

```
public void transferToSaving(double amount) throws
                                InsufficientBalanceException {
   checkingBalance -= amount;
   savingBalance += amount;
   try {
        updateChecking(checkingBalance);
        if (checkingBalance < 0.00) {
        context.setRollbackOnly();
        throw new InsufficientBalanceException();
        }
        updateSaving(savingBalance);
   } catch (SQLException ex) {
        throw new EJBException
        ("Transaction failed due to SQLException: "+
        ex.getMessage());
   }
}
```

Application exception : explicit roll back

System exception : automatic roll back

Java 5 Annotations Example

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)

public class OrderManagerBean {

@Resource
private SessionContext context;
...
@TransactionAttribute(TransactionAttributeType.REQUIRED)
public void placeOrder(Item item, Customer customer){
   try {
        if (!bidsExisting(item)){
        validateCredit(customer);
        chargeCustomer(customer, item);
        removeItemFromBidding(item);
        }
   } catch (CreditValidationException cve) {
        context.setRollbackOnly();
   } catch (CreditProcessingException cpe){
        context.setRollbackOnly();
   } catch (DatabaseException de) {
        context.setRollbackOnly();
   }
}

}
```

@ApplicationException

```
public void placeOrder(Item item, Customer customer) throws
   CreditValidationException,
   CreditProcessingException, DatabaseException {

   if (!bidsExisting(item)){
       validateCredit(customer);
       chargeCustomer(customer, item);
       removeItemFromBidding(item);
   }
}
...
@ApplicationException(rollback=true)
public class CreditValidationException extends Exception {
...
@ApplicationException(rollback=true)
public class CreditProcessingException extends Exception {
...
@ApplicationException(rollback=false)
public class DatabaseException extends RuntimeException {
...
```

J2EE Security Terminology (1)

A ***principal*** is "something" that can be authenticated

- For example, a user or a server

Each principal has an associated set of ***security attributes***

- Used to identify which resources the principal can access

A principal is identified using ***credentials***

- A credential contains or references security attributes
- Credentials are acquired via authentication
- Credentials can also be acquired through delegation from another principal

J2EE Security Terminology (2)

Group: A set of authenticated users, defined in the Application Server.

Role: An abstract name for the permission to access a particular set of resources in an application.

- A role can be compared to a key that can open a lock.
- Many people might have a copy of the key. The lock doesn't care who you are, only that you have the right key.

Two possible ways

Declarative security: defined using deployment descriptors

- Includes definition of security roles, access control rules and authentication requirements
- Mapped by the application deployer to the specific runtime environment

Programmatic security: explicit use of security APIs by application code

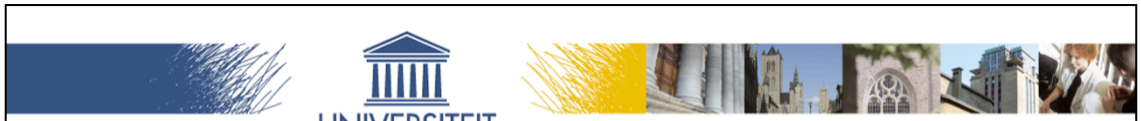
- Provides increased flexibility
  - e.g., the same method can function differently for different principals
- Key methods: `getCallerPrincipal` and `isCallerInRole`, defined in `EJBContext`.

Declarative example

```
@DeclareRoles("BIDDER", "CSR", "ADMIN")
@Stateless
public class BidManagerBean implements BidManager {

@RolesAllowed("CSR, ADMIN")
public void cancelBid(Bid bid, Item item) {...}

@PermitAll
public List<Bid> getBids(Item item) {...}
}
```

Programmatic example

```
@Stateless
public class BidManagerBean implements BidManager {

@Resource SessionContext context;
...
public void cancelBid(Bid bid, Item item) {
   if (!context.isCallerInRole("CSR")) {
        throw new SecurityException("No permissions to
              cancel bid");
   }
   ...
}
...
}
```



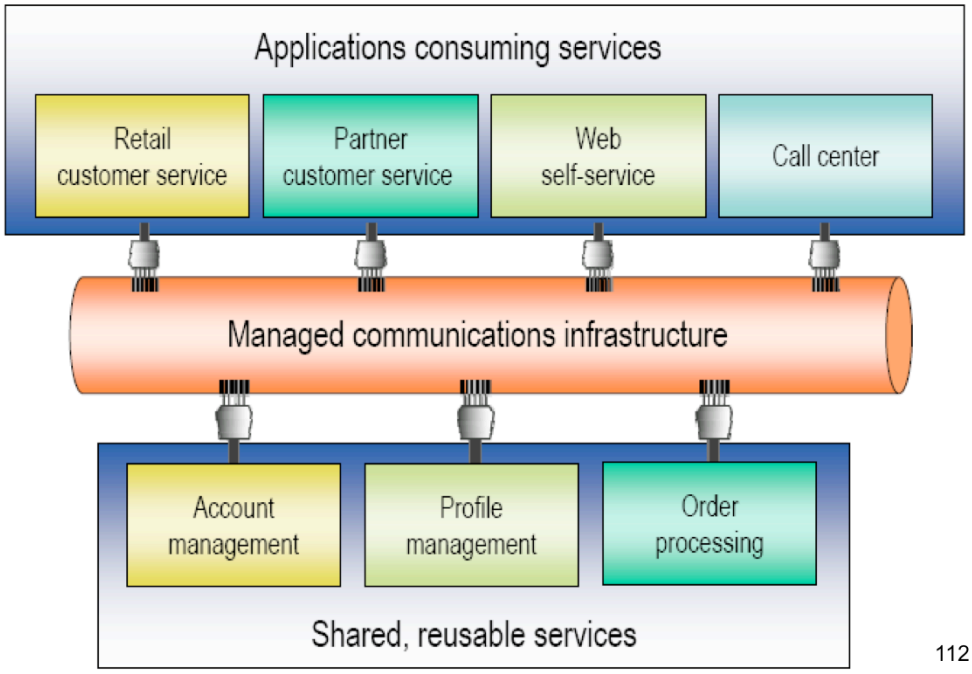
Service Oriented Architectures

Outline

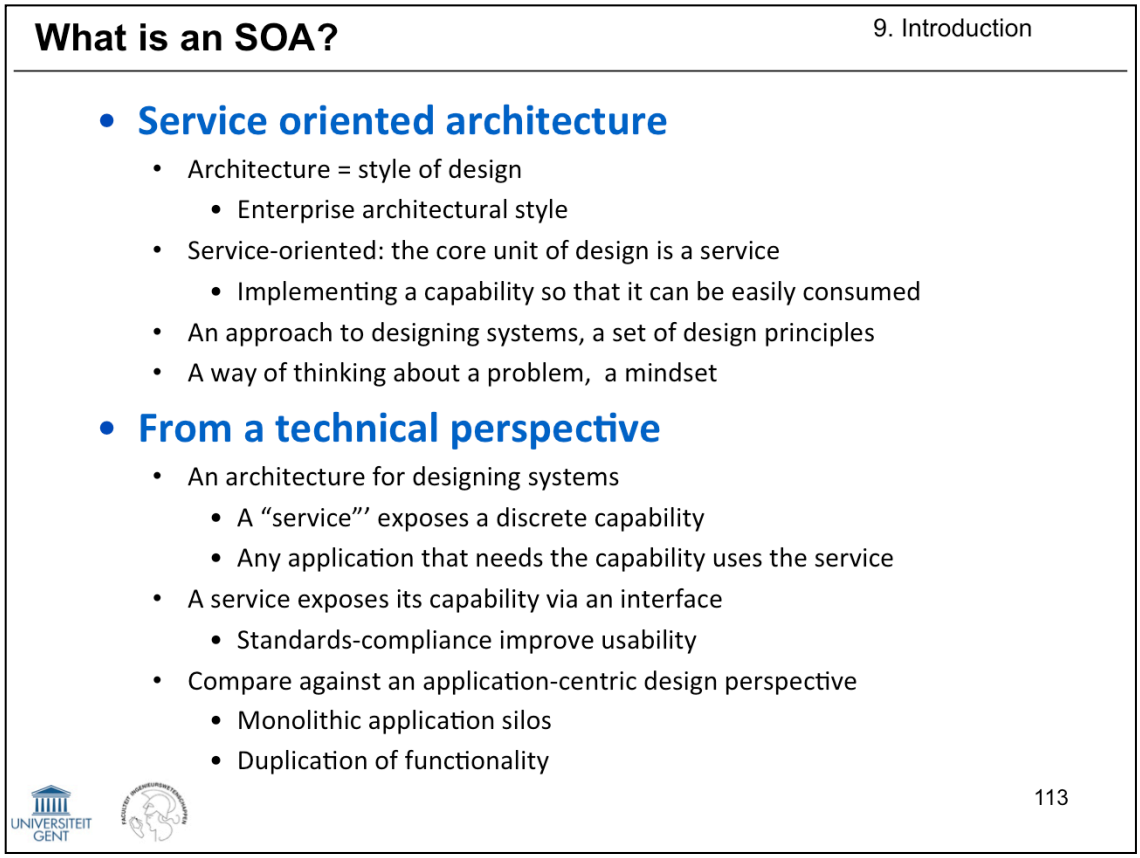
9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12. Web Service Orchestration and Choreography
13. Software Integration
    - eTOM, ITIL

What is an SOA?

- **Shared, reusable, loosely coupled services**



One of the key features of an SOA is the ability to support shared, cross-application reusable services. Application designers can use the functionality they need from a variety of loosely-coupled service building blocks and hence rapidly and robustly create new applications or application features.



A SOA (Service Oriented Architecture) is first and foremost an architecture, which means it is a design style tuned for enterprises. The other key ingredient of an SOA is the fact that it is service oriented: the core building block of an SOA design is a service. These service building blocks implement capabilities that can be easily (automically) consumed.

Generalised we can state that the SOA idea is an approach to designing enterprise systems, it gives the designer a set of design principles by which a problem can be solved.

Seen from a technical perspective, SOA is, as stated, an architecture for designing systems, in which the service building blocks expose certain capabilities. Any application that wants to use those capabilities can ask the service to deliver them. Services themselves expose their capabilities via a service interface (which is standards-compliant to improve interoperability).

When compared against application-centric design, the application-centric approach leads to monolithic applications with a lot of duplicated functionality.

SOA benefits

- **Measurable benefits**
  - Improvement of the quality of applications
    - Outsourcing non-business functionality to knowledgeable service creators
    - Increase productivity, efficiency, effectiveness, satisfaction
  - Reduction of time-to-market
    - Deliver new features, new capabilities faster
  - Increase of ease of doing business
    - Make it easier to integrate with customers, partners and suppliers
  - Reduction of costs

There are plenty of benefits of employing a SOA model:

First and foremost SOA can improve the quality of applications: by combining modular service block functionality, businesses can focus on the functionality they are specialised in, and leave other functionality to be implemented by specialised service builders.

SOA can reduce the time to market a new application, as certain (more trivial) service building blocks can either be reused from existing applications or bought from other service developers.

SOA also increases the ease of doing business, as due to the adoption of standards it becomes easy to integrate suppliers' services in products (i.e. less time spent figuring out interworking mechanics), or to deliver services yourself to customers.

Due to outsourcing / service-provider competition, the development and maintenance costs of business-needed functionality can be lowered. An additional benefit is that new applications do not have to be developed from a clean slate, but can instead count on a toolbox of existing services from which development can be started.

SOA risks

- **Risks**
  - SOA is hard
  - Disruptive to the status quo
    - Fundamental changes required
  - Requires a long term commitment
    - Modest return in the near term
    - Significant return in the long term
  - Many SOA initiatives will fail spectacularly
    - Derailed by cultural issues, not technology
      - Deliver on time
      - Cost containment
  - Systems may become more brittle and inflexible than before

Along with the benefits come plenty of risks though. One of these risks is that SOA is a difficult concept to execute in a good way: fundamental changes are required in the application development cycle, concerning viewing applications no longer as individual-standing containers, but instead as a collection of self-containing services with the core 'application-glue' welded between these services.

For businesses SOA requires a long term commitment, as the return in the near term will be modest at best, but in the long term, if done right, significant investment returns can be expected.

A lot of SOA initiatives are bound to fail, mostly due to cultural issues, and not due to the state of SOA technology. Projects nowadays are expected to be delivered on-time, within a certain cost-frame. Mostly these limitations do not take into account the fact that if projects are built up in a SOA way, the hence developed services could be reused in future projects (lowering those projects' development time and focusing more on innovative features).

If SOA is not done in a correct fashion then there is a profound risk of making systems even more brittle than before, as, when for instance relying on external parties' services, any wrongful (i.e. non-announced functionality breaking change) change to this external service can leave the whole system in distress.

SOA requires changes

- **Fundamental changes required**
  - Design-wise
    - Different design focus (service rather than application)
    - Designing reusable services takes more time
    - Requires much more collaboration
  - Accounting
    - Who pays for initial service development?
    - How does one re-gain those expenses?
  - Control issues
    - Hard dependencies on services that you do not control
  - Operations issues
    - New applications may impact performance of existing apps

From looking at the risks it has already become clear that, in order to have SOA succeed, some changes are required.

One of these changes is that design needs to focus more on the individual services rather than on the whole application. This design of reusable service building blocks requires more time and much more collaboration (the to-be-developed service building block needs to be able to work ‘on its own’ and provide its’ functionality to a variety of service consumers).

Cost-wise a business needs to look at how the initial (and rather time-consuming) service developments will be budgetted and how it will be able to regain those expences.

Using external (i.e. not under company control) services also requires hard dependencies to be negotiated regarding the functionality and behavior that those services will exhibit (and will keep exhibiting over time).

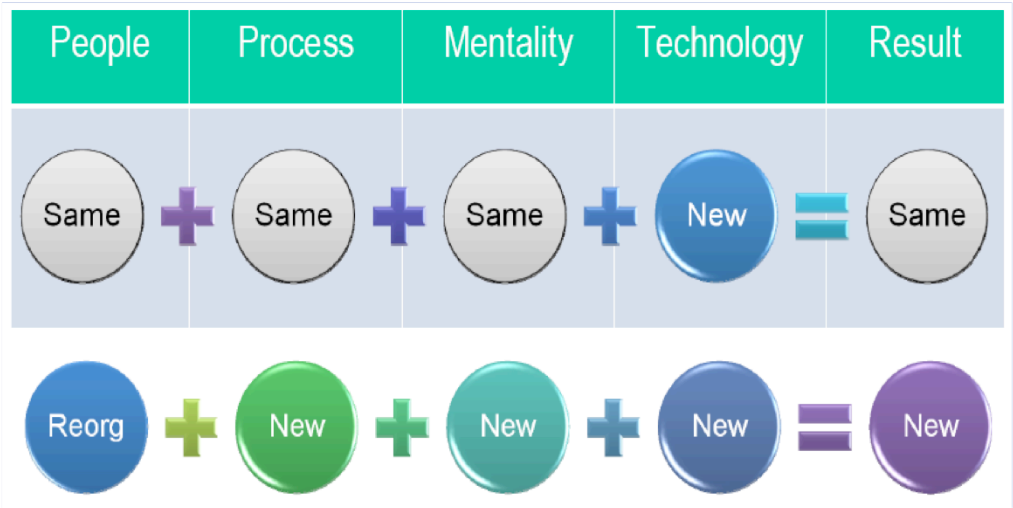
Special care also has to be taken when deploying new applications that make use of existing services, as the added load on those existing services can lead to worse performance for existing applications.

SOA planning and governance

- **SOA requires planning and governance**
  - Where do you start?
  - How do you build services?
  - Do you build the right services?
  - Are your services reusable?
  - Can people find the services?
  - Who is using the services?
  - Can you find the root cause of a problem?
  - Who will be affected if you make this change?

Moving to SOA mainly involves a big deal of careful planning and governance. How will one make the good choices regarding which functionality will be contained in services (so that they are not trivial on the one side and not bloated on the other)? How will services be made reusable across applications, whether it be in-company or outsourced? Are people able to find the repository of already existing services? Who is using your services? What if problems occur, can one easily backtrace to the root of a problem? If you make changes to existing services, how will it affect existing applications that make use of that particular service.

New technology != change



A new technology alone is not sufficient to provide inherent change, instead, as was mentioned, both the process and mentality behind developing IT projects should change. The service oriented process / mentality and coupled technology can make it possible to achieve new results.

SOA enabling technologies

- **SOA is about design not technology**
  - Any technology can be used to build a service
    - WS-*, CORBA, COM, etc
  - Technology will influence the power, flexibility, manageability and resilience of a SOA system
  - Pervasive standards-based technology = wider reach
    - Standard data formats (e.g. XML)
    - Standard protocols (e.g. HTTP, SOAP)
    - Standard description languages (e.g. XML Schema, WSDL)
  - Isn't WS-* required?
    - SOA is about system design, not technology
    - 10 years from now, WS-* may be like CORBA
    - Today: one of the best options available
      - Pervasive: widely adopted
      - Good tooling and frameworks
      - Good interoperability

SOA is not about a specific technology but rather about a design view. The chosen technology will however influence how powerful, flexible, manageable and resilient to changes a SOA-based system will be.

The choice for a standards-based technology will allow SOA developers to have a wider reach (making it easier to reuse their services for other applications).

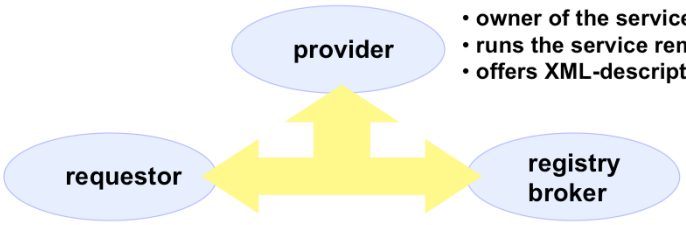
For SOA, it is not needed to use the various WS-* standards, but today the WS-* standards are one of the best options for creating SOA, as they have been widely adopted and have good tooling available.

Service-Oriented Architectures

Vision

Services should be dynamically discoverable,
Separation between service description and implementation
Assemble application “ad hoc” based on discovered services

“Service-oriented architectures”



- owner of the service
- runs the service remotely
- offers XML-description of service

- discovers and invokes service
- can be person/application/service

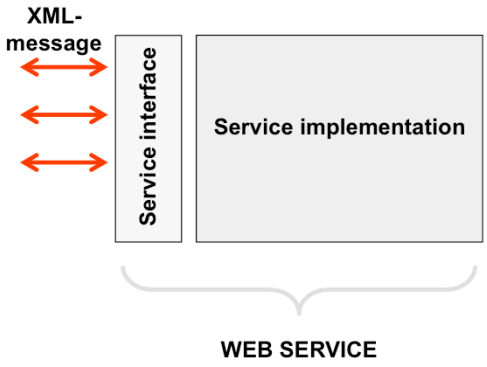
- manages info repository on providers and services
  - white pages : business info on provider (name , description, …)
  - green pages : info how to use the service (policies, software bindings, …)
  - yellow pages : key-word based search engine for services

Outline

9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12. Web Service Orchestration and Choreography
13. Software Integration
    - eTOM, ITIL

Web Service

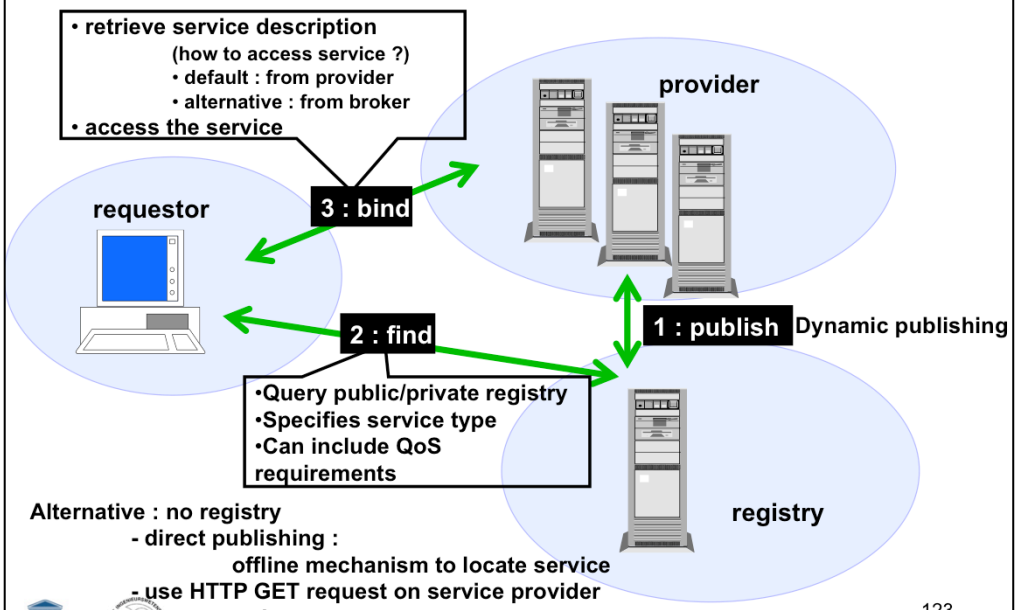
"A Web service is a software application identified by a URI whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts, and supports direct interactions with other software applications using XML-based messages via Internet-based protocols."



Service implementation :
- any application
  - any programming language
  - any platform
- remotely available through web service provider

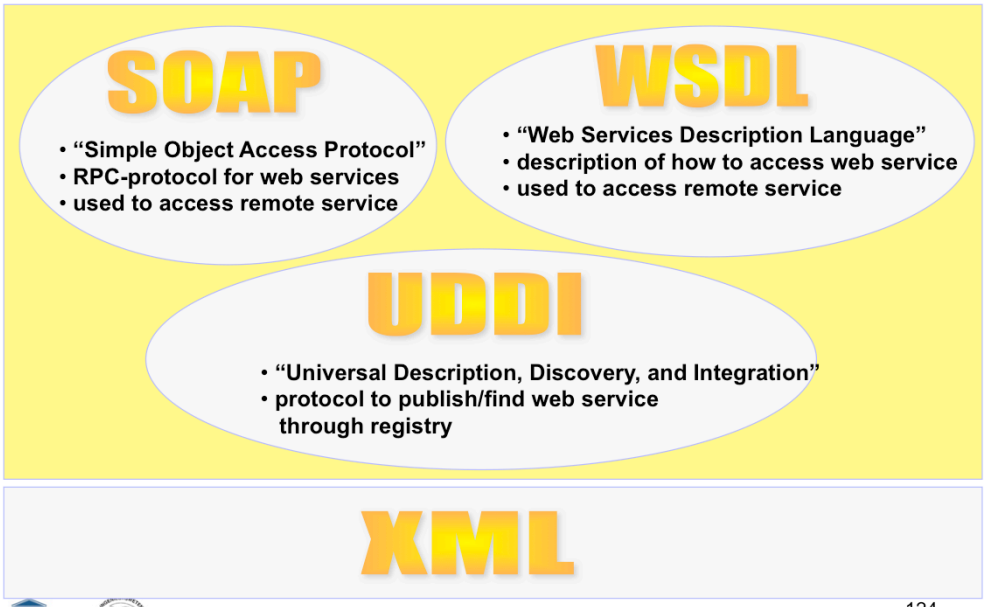
Service definition/interface :
- defines the service
- in XML-syntax
- not necessarily on same machine as service implementation
- includes :
  - data types involved
  - supported method calls
  - network location

Service-Oriented Architectures

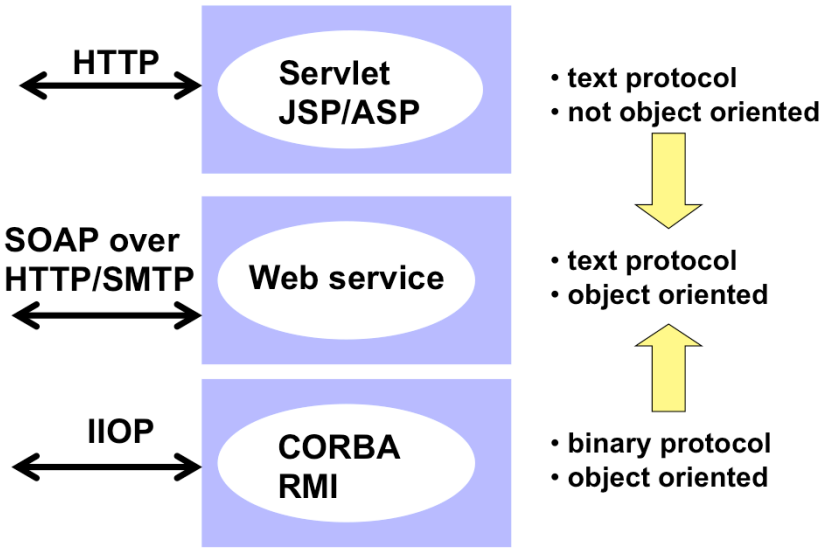


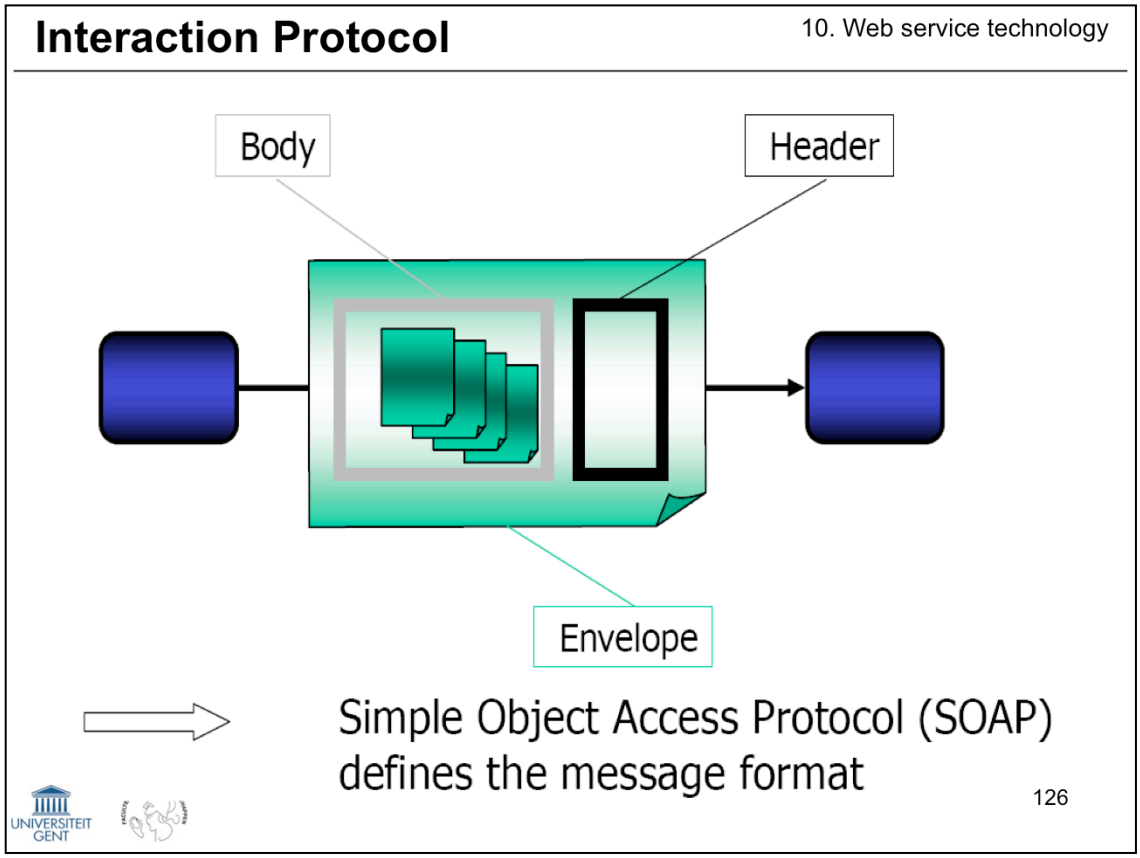
Alternative : no registry
- direct publishing :
offline mechanism to locate service
- use HTTP GET request on service provider
webserver

Web Service Technologies



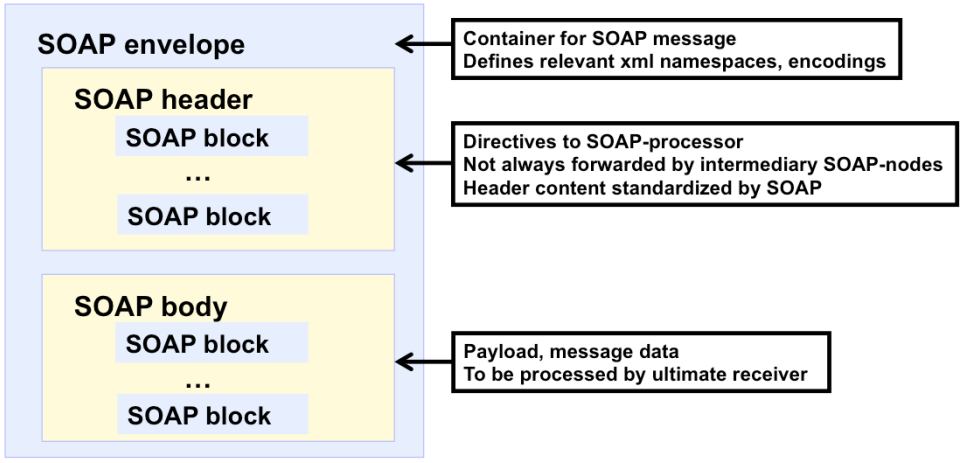
Web Service Technology





XML being the data format of choice for web service data exchanges, we still need an interaction protocol between client / service. This interaction protocol has been defined to be SOAP (Simple Object Access Protocol), which defines the message format, consisting of a SOAP envelope containing a header and message body. Note that other interaction protocols exist.

SOAP messages



SOAP-processor = environment receiving/processing/forwarding SOAP-message
(e.g. J2EE application server, .NET server, …)

SOAP message : request example

SOAP RPC request

```
POST /InStock HTTP/1.1
Host: www.stock.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPrice>
      <m:StockName>IBM</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

HTTP protocol binding

Specify namespaces And encoding rules

RPC-specification

stock.GetStockPrice(“IBM”)

SOAP message : reply example

SOAP RPC reply

```
HTTP/1.1 200 OK
Content-Type: application/soap; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

  <soap:Body xmlns:m="http://www.stock.org/stock">
    <m:GetStockPriceResponse>
      <m:Price>34.5</m:Price>
    </m:GetStockPriceResponse>
  </soap:Body>
</soap:Envelope>
```

“return 34.5”

Web Services Description Language

- XML grammar to specify collection of "access end points"
  (1 URL specifies a single access end point)
- designed to automate application-to-application interaction (or B2B interaction)
- defines the *communication protocol* to be used at runtime
  - message format
  - methods to be invoked
  - parameter lists, return types
  - …
- WSDL descriptions can be automatically generated for existing code
- stub classes can be generated from WSDL descriptions

WSDL1.1 : document structure


```
<definitions>
  <import>*
  <types>
        <schema></schema>*
  </types>
  <message>*
        <part></part>*
  </message>
  <portType>*
        <operation>*
              <input></input>
              <output></output>
              <fault></fault>*
        </operation>
  </portType>
  <binding>*
        <operation>*
              <input></input>
              <output></output>
        </operation>
  </binding>
  <service>*
        <port></port>*
  </service>
</definitions>
```

<definitions>
- container for service description
- global declaration of
  document scope namespaces

<import>
- similar to java-import
  (modularization of WSDL docs)
- include of namespaces (not file itself !)

<types>
- contains definition of all datatypes used in messages
- currently : XSD (XML Schema Definitions)

<message>
- defines message structure
  (for each part : type and name are defined)

<portType>
- specifies legal operations for web service endpoint (=group of actions)
- <operation> : method definition specifies
  - method name
  - input message
  - output message
  - error message

WSDL1.1 : document structure


```
<definitions>
  <import>*
  <types>
        <schema></schema>*
  </types>
  <message>*
        <part></part>*
  </message>
  <PortType>*
        <operation>*
              <input></input>
              <output></output>
              <fault></fault>*
        </operation>
  </PortType>
  <binding>*
        <operation>*
              <input></input>
              <output></output>
        </operation>
  <binding>
  <service>*
        <port></port>*
  </service>
</definitions>
```

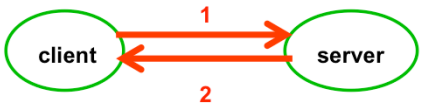
<binding>
- **concrete protocol & data format spec for portType**
- **standard soap-bindings available**

<service>
- **defines URL for service endpoint**
- **<port> = single access point for webservice**
- **each port specifies**
  - **name**
  - **binding info**
  - **URL of webservice (<soap:address> element)**

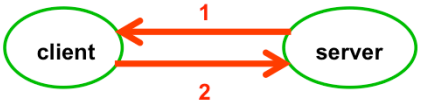
WSDL1.1 : supported interactions

<operation>

Request – response
<input> before <output>



Solicit - response
<output> before <input>



One-way (asynchronous)
only <input>

Notification
only <output>

WSDL1.1: different styles

RPC/encoded

RPC/literal

Document/encoded (not used)

Document/literal

RPC/encoded

public void myMethod(int x, float y);

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- ... -->
```

```
<soap:envelope>
  <soap:body>
    <myMethod>
      <x xsi:type="xsd:int">5</x>
      <y xsi:type="xsd:float">5.0</y>
    </myMethod>
  </soap:body>
</soap:envelope>
```

-XSI types usually overhead

- no easy validation

- not WS-I compliant (WS-I defines interoperability rules)

RPC/literal

```
public void myMethod(int x, float y);
```

```
<message name="myMethodRequest">
    <part name="x" type="xsd:int"/>
    <part name="y" type="xsd:float"/>;
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- ... -->
```

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

↓

- no overhead from XSI types
- still no easy validation
- WS-I compliant

Document/literal

public void myMethod(int x, float y);

```
<types>
    <schema>
        <element name="xElement"
   type="xsd:int"/>
        <element name="yElement"
   type="xsd:float"/>
    </schema>
</types>

<message name="myMethodRequest">
    <part name="x" element="xElement"/>
    <part name="y" element="yElement"/>
</message>
<message name="empty"/>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- ... -->
```

```
<soap:envelope>
    <soap:body>
        <xElement>5</xElement>
      <yElement>5.0</yElement>
    </soap:body>
</soap:envelope>
```

- no operation name (difficult dispatching)

- easy validation (soap:body can be defined in a schema)

- not WS-I compliant

Document/literal wrapped

public void myMethod(int x, float y);

```
<types>
    <schema>
        <element name="myMethod">
            <complexType>
                <sequence>
                    <element name="x" type="xsd:int"/>
                    <element name="y" type="xsd:float"/>
                </sequence>
            </complexType>
        </element>
        <element name="myMethodResponse">
            <complexType/>
        </element>
    </schema>
</types>
<message name="myMethodRequest">
    <part name="parameters" element="myMethod"/>
</message>
<message name="empty">
    <part name="parameters" element="myMethodResponse"/>
</message>

<portType name="PT">
    <operation name="myMethod">
        <input message="myMethodRequest"/>
        <output message="empty"/>
    </operation>
</portType>

<binding .../>
<!-- ... -->
```

```
<soap:envelope>
    <soap:body>
        <myMethod>
            <x>5</x>
            <y>5.0</y>
        </myMethod>
    </soap:body>
</soap:envelope>
```

↓

- operation name is available
- easy validation (soap:body can be defined in a schema)
- WS-I compliant

Summary WSDL styles

Use document/literal wrapped

- except: when overloaded operations

```
public void myMethod(int x, float y);
public void myMethod(int x);
```

- then: use document/literal unwrapped
- however: when several methods have same signature

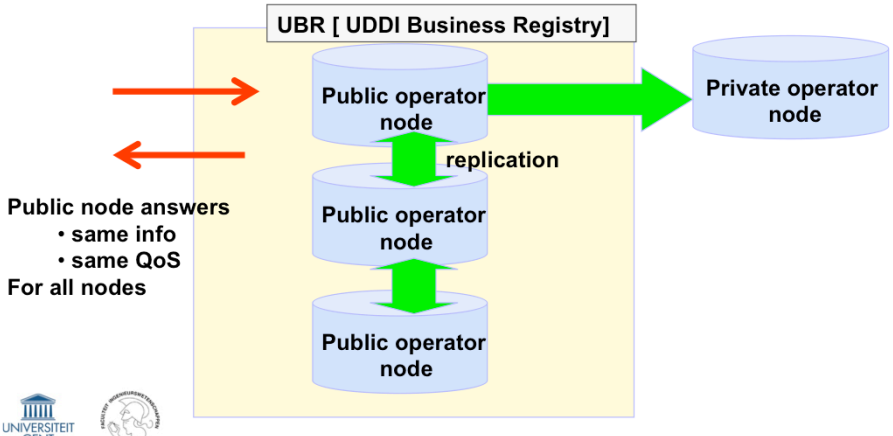
```
public void myMethod(int x, float y);
public void myMethod(int x);
public void myOtherMethod(int x, float y);
```

- then: use RPC/literal
- in case of data graphs (e.g. binary trees), literal style is not an option, prefer RPC/encoded

Ref: http://www.ibm.com/developerworks/webservices/library/ws-whichwsdl/

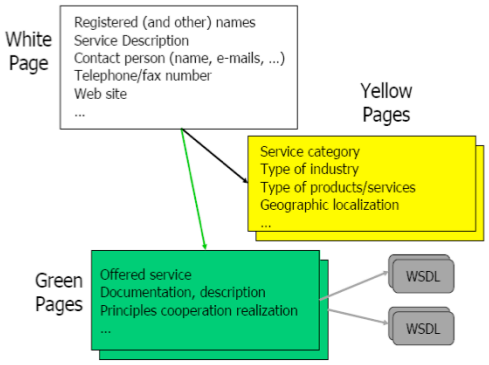
Universal Description, Discovery, and Integration

- = standardized method to publish and discover information about web services
- DNS-like functionality (and organization)
- currently not very popular



How to offer and find services?

- **Universal Description, Discovery and Integration (UDDI)**
- **Standardized publishing, advertisement**
- **Extended name server, describing interface and properties**
- **XML Descriptor**
  - White page: service name, description, contact information
  - Yellow page: semantics (based on standard taxonomy)
  - Green page: technical specification (URL, WSDL, etc.)
- **Logically central, physically distributed database**



How can we now offer our web services to the world and / or find web services that deliver us the functionality that we need? The answer lies in the UDDI standard: a standardized way of publishing / advertising web services. UDDI can be seen as an extended name server that XML-describes web service interfaces and properties. The descriptor consists of three parts: the white pages, yellow pages and green pages.

The white pages hold basic contact information regarding the web service (who develops it, what is the name, how can we contact the developers, etc.). The yellow pages cover the semantics of the web service and the green pages cover the technical aspects: the URL, interface (WSDL), documentation etc. This service information repository can be seen as a logically central but physically distributed data base.

UDDI: the reality

- **UDDI is, due to non-common usage, slowly becoming an outdated standard**
- **Web Services are increasingly being used in Business-2-Business (B2B) / Enterprise Application Integration (EAI) scenarios**
  - Trusted environments
  - "Certainty" about service support
- **No widespread 'service' market**
- **Not so great as a public database (Microsoft, IBM, SAP closed their test UDDI databases)**

The reality of the UDDI standard is that it is slowly becoming outdated, due to in effect almost never being used. Currently the general business public has no need for a widespread 'service' market, but instead opts for trusted Business-2-Business web service use scenarios, where parties trust and know each other.

Microsoft, IBM and SAP even closed their UDDI test databases.

Alternative to UDDI?

- **UDDI provides discovery for services that are always connected to the network.**
  - need a discovery system for sometimes connected services.
- **UDDI has to handle with out-of-date service entry information**
  - Need a discovery system for dynamically updated entries.
- **UDDI provides discovery for only registered services**
  - Need a discovery for services not exist in any central registry.
- **UDDI provides a central registry**
  - Need a discovery system which performs on distributed registries on ad hoc and managed networks

With the downsides of UDDI in-mind, is there an alternative to this standard?

More specifically we would like to find a technology that can discover services that are only connected from time-to-time, that dynamically updates service information and that finds services that aren't registered to a central repository.

WS-Discovery

- **Defines a multicast protocol to locate services**
- **allows dynamic discovery of services in ad-hoc and managed networks**
  - discovery of temporarily-connected services providing interface to portable devices such as hand-helds, pocket pc, etc…
- **enables discovery of services by type and within scope**
- **scales to a large number of endpoints, by defining a multicast suppression behavior if a service registry (discovery proxy) is available on the network**
- **Supported by Microsoft, BEA, Intel, Canon**
- **No internet-scale discovery, aimed at local networks**

This alternative exists under the WS-Discovery standard. WS-Discovery defines a multicast protocol to locate services. This multicast protocol is able to dynamically discover services in both ad-hoc and managed networks. It allows discovery of services by type (i.e. can specify what the services must adhere to, after which only services matching those specifications will be searched). The multicast protocol used can scale to a large number of endpoints as it automatically suppresses multicasting when a service registry is available on a network.

WS-Discovery is supported by a large number of high-profile vendors. It does not allow internet-scale discovery of services.

WS-Addressing

WS-Addressing defines standard ways to route a message over multiple transports or direct a response to a third party.

- Without WS-Addressing: a standard SOAP request is sent over HTTP, the URI of the HTTP request serves as the message's destination. The message response is packaged in the HTTP response and received by the client over the HTTP connection.

For example, a client application might send a request over JMS and ask to receive the response through e-mail or SMS.

To enable these kinds of applications, WS-Addressing incorporates delivery, reply-to, and fault handler addressing information into a SOAP envelope.

WS-Addressing example

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
                xmlns:wsa="http://www.w3.org/2004/12/addressing">
    <S:Header>
        <wsa:MessageID>
            http://...
        </wsa:MessageID>
        <wsa:ReplyTo>
            <wsa:Address>http://…</wsa:Address>
        </wsa:ReplyTo>
        <wsa:FaultTo>
            <wsa:Address>http://…</wsa:Address>
        </wsa:FaultTo>
        <wsa:To>http://…</wsa:To>
        <wsa:Action>http://…</wsa:Action>
    </S:Header>
    <S:Body>
         <!-- The message body of the SOAP request -->
    </S:Body>
</S:Envelope>
```

WS-Addressing Endpoint references

```
<wsa:EndpointReference xmlns:wsa="..." xmlns:example="...">
   <wsa:Address>http://example.com/weather</wsa:Address>
   <wsa:ReferenceProperties>
       <example:ServiceLevel>Basic</example:ServiceLevel>
   </wsa:ReferenceProperties>
   <wsa:ReferenceParameters>
       <example:CityCode>NYC</example:CityCode>
   </wsa:ReferenceParameters>
</wsa:EndpointReference>
```

WSDL2.0

W3C working draft since 2004, recommendation since july 2007
rarely used for the moment: WSDL 1.1 still widely used

Main differences with WSDL1.0
- support for WS-Addressing
- interface type instead of port type
- no operation overloading allowed

Outline

9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12. Web Service Orchestration and Choreography
13. Software Integration
    - eTOM, ITIL

JAX WS

Java EE 5 provides a robust platform on which web services can be built and deployed

Web service can be built either as:

- Option 1: regular Java class
- Option 2: EJB 3 stateless session bean

Option 2 allows for use of declarative transactions and security.
Advantage Option 1: can be run in a web container, no need for application container.
Both options can use annotations, definition of life cycle methods.
Option 1 requires annotation processing in an external Annotation Processing Tool (APT)

Ref: http://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html

EJB as a Web Service example


```
@WebService
@SOAPBinding(style = SOAPBinding.Style.DOCUMENT)

@Stateless
public class eBayBean implements eBayInterface {

@PersistenceContext private EntityManager em;
   public eBayBean() {
   }
   @WebMethod
   @WebResult(name = "bidNumber")

   public Long addBid(
   @WebParam(name = "User") String userId,
   @WebParam(name = "Item") Long itemId,
   @WebParam(name = "Price") Double bidPrice) {
        return persistBid(userId, itemId, bidPrice);
   }

   private Long persistBid(String userId, Long itemId, Double bidPrice)
   {
   }
}
```

@WebService annotation options

```
@ WebService {
   String name() default "";
   String targetNamespace() default "";
   String serviceName() default "";
   String wsdlLocation() default "";
   String endpointInterface() default "";
   String portName() default "";
};
```

@SOAPBinding annotation options

```
@ SOAPBinding {
   public enum Style {
       DOCUMENT,
       RPC
   };
   public enum Use {
       LITERAL,
       ENCODED
   };
   public enum ParameterStyle {
       BARE,
       WRAPPED
   };

   Style style() default Style.DOCUMENT;
   Use use() default Use.LITERAL;
   ParameterStyle parameterStyle() default
       ParameterStyle.WRAPPED;
}
```

@WebMethod annotation


```
@WebService
  @Stateless
  public class eBayBean {
      public Long addBid(..) {
      }
  @WebMethod(exclude = "true")
  public Long persistBid(..) {
  }
}
```

@WebMethod annotation options

```
@WebMethod {
  String operationName() default "";
  String action() default "" ;
  boolean exclude() default false;
};
```

@WebParam annotation

```
@WebMethod
public Long addBid(
@WebParam(name = "user", mode = WebParam.Mode.IN)
  String userId, ...) {

...

}
```

@WebParam annotation options

```
@WebParam {
  public enum Mode { IN, OUT, INOUT };
  String name() default "";
  String targetNamespace() default "";
  Mode mode() default Mode.IN;
  boolean header() default false;
  String partName() default "";
};
```

@WebResult annotation + options


```
@WebMethod
@WebResult(name = "bidNumber")
  public Long addBid(...){}
```

Options:

```
@WebResult {
  String name() default "return";
  String targetNamespace() default "";
  boolean header() default false;
  String partName() default "";
};
```

Accessing a web service from an EJB

```
import javax.xml.ws.WebServiceRef ;

@WebServiceRef(wsdlLocation=
"http://localhost:8080/eBayService/eBayBean?WSDL")
private static eBayService ebs;

public static void main(String [] args) {
   try {
        eBayBean ebay = ebs.geteBayBeanPort();
        System.out.println("Bid Successful, BidId Received is:“
               +ebay.addBid(“test", 1001, 185,0 ));
   } catch (Exception ex) {
        ex.printStackTrace();
   }
}
```

EJB as a web service client


```
@Stateless
public class AmazonOrderBean implements AmazonOrder {
@WebServiceRef(AmazonDeliveryService.class)
private AmazonDeliverySEI deliveryService;

public String checkOrderDeliverStatus(String shipId) {
   ...
   String deliveryStatus =
       deliveryService.checkDeliveryStatus(shipId);
   ...
}
}
```

NetBeans support for JAX-WS:
http://www.netbeans.org/kb/docs/websvc/jax-ws.html

Outline

9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12. Web Service Orchestration and Choreography
13. Software Integration
    - eTOM - ITIL

What is the problem?

- **Web service composition**
  - Bringing together existing web services to build more complex ones
  - Orchestration e.g. WS-BPEL
- **Consistency of composition**
  - Guaranteeing consistency between behaviour delivered by Web Services and expected composition
  - Choreography e.g. WS-CDL

In this section we focus on how to build (multi-partner) business processes using different (multi-partner distributed) web services.

One of the methods used to do so can be web service orchestration (by means of for instance WS-BPEL), which aims to build complex new services by using existing web services in a controlled workflow (and coordinated by a single coordinator).

Another technology, web service choreography (by means of the WS-CDL standard) aims to look at the consistency of composing web services from multiple independent parties into a business process (with no central coordinator).

Choreography versus Orchestration

Choreography	Orchestration
Distributed environment	Centralized environment
All participants are active and independent	1 active & multiple passive participants
Collaboration	Composition
Interaction based *(observable behavior)*	Action based

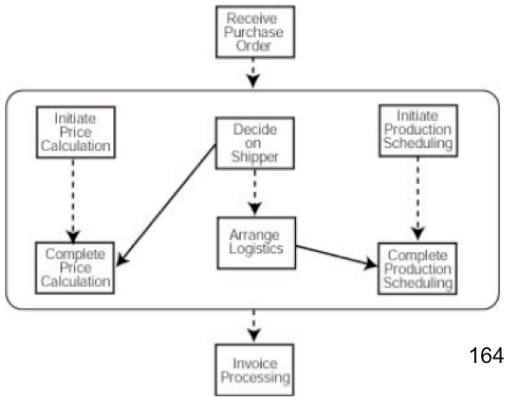
The differences between a web service choreography and a web service orchestration can be seen on this slide:

A web service choreography operates in a fully distributed environment, with no central coordinator, while an orchestration needs a central coordinator to control the workflow execution. In a choreography all participants are active and behaving independently from each other, while in an orchestration there is one active participant, controlling the total workflow behavior, and multiple passive participants, waiting for instructions by the active party.

Also, like in a dance, a choreograpy is fully interaction based (what matters is the total system behavior), while in an orchestration what matters is the actions undertaken as directed to by the coordinator of the orchestration.

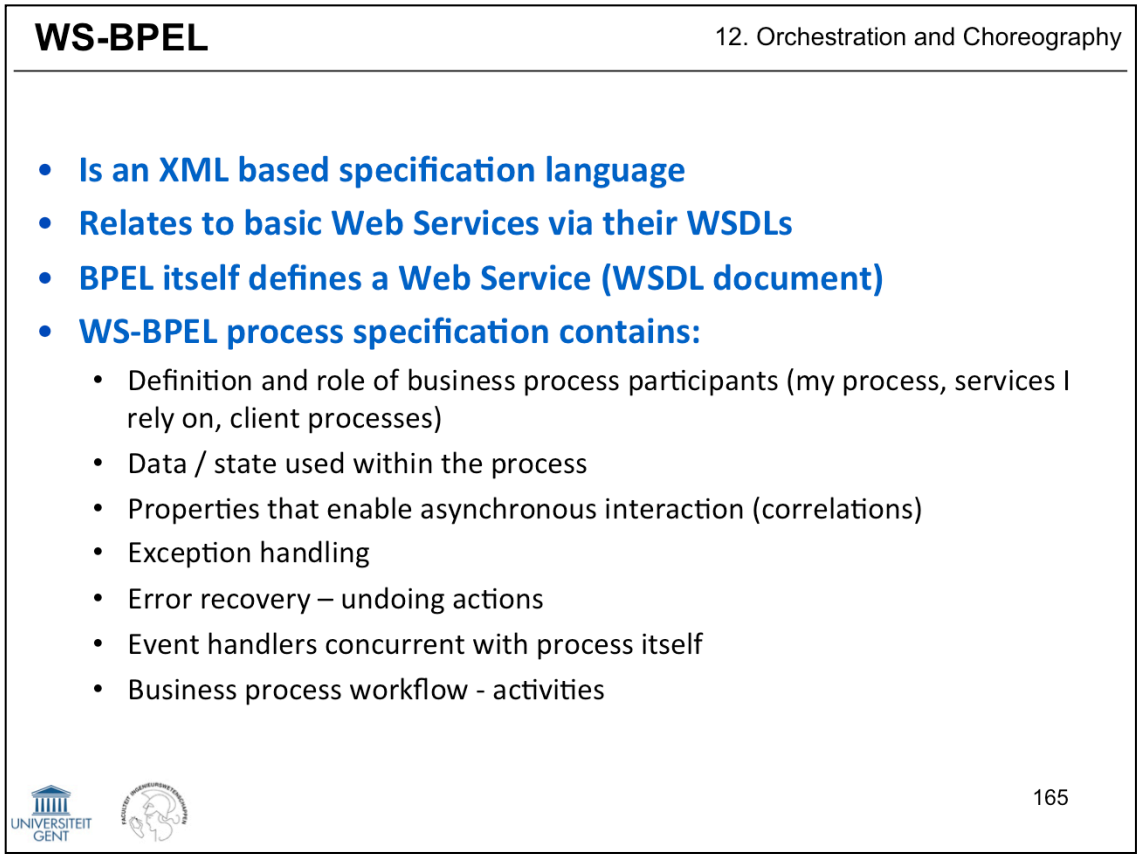
Business Process Execution Language

- Workflow definition language for describing business processes
- WS-BPEL now, before BPEL4WS
- Convergence of XLANG (Microsoft) and WSFL (IBM)
- 1st version released by IBM, Microsoft and BEA in august 2002
- Version 2.0 standard since April 2007
- Example workflow



BPEL is the standard worflow definition language for describing business processes (or so called workflows) with web services. Before it was called BPEL4WS, but now, to be better aligned with the other WS-* standards, it has been renamed to WS-BPEL.

In the example we see a business workflow stating that when a purchase order is recieved, three activities need to be started: initiating a price calculation, deciding on shipper and initiating a production schedule. Once the shipper has been decided the price calculation can be completed, and logicstics can be arranged. When price calculation and production scheduling activities are finished, invoice processing can begin.



BPEL is an XML-based specification language that uses Web Services' WSDL to compose new business workflows. The BPEL workflow as a whole can itself be seen as a (new) Web Service (exposing a WSDL document).

The WS-BPEL specification contains a.o.:

The definition and role of business process participants (the actual business process, the web services that are needed to implement this business process, client processes), the data (and associated state) of the process, properties that enable asynchronous calls (so called correlations).

Exception handling, error recovery and event handlers are all possible in BPEL, as are general business process workflow activities (joins, branches, etc.)

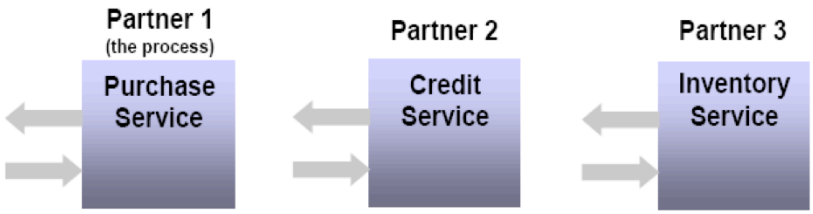
BPEL scenario structure


```
<process>
  <!– Definition and roles of process participants -->
  <partnerLinks> ... </partnerLinks>
  <!- Data/state used within the process -->
  <variables> ... </variables>
  <!- Properties that enable asynchronous interaction -->
  <correlationSets> ... </correlationSets>
  <!- Exception handling -->
  <faultHandlers> ... </faultHandlers>
  <!- Error recovery – undoing actions -->
  <compensationHandlers> ... </
compensationHandlers>
  <!- Concurrent events with process itself -->
  <eventHandlers> ... </eventHandlers>
  <!- Business process flow -->
  (activities)*
</process>
```


The pictured XML shows the structure of a BPEL.

BPEL PartnerLink

- **All parties the BPEL interacts with are defined by partner links**
  - Links to all the different WSDL-defined Web Services that a BPEL process instance interacts with
  - Invoked partner links: BPEL process invokes Web Service
  - Client partner link: WSDL-defined Web Services that can invoke a BPEL process
    - At least one (start client)
  - A Web Service can have both roles (WS-invokes BPEL and BPEL-invokes-WS)



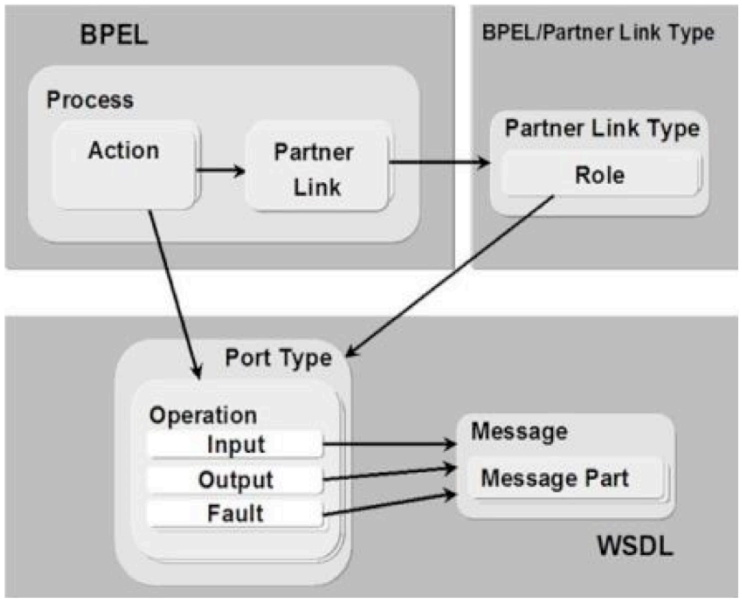
In the partner link section of the BPEL the different partner links are declared. Partners in this case are either web services that are invoked by the BPEL process, or links to web services which can invoke the BPEL process. The latter can be the start-client of the BPEL process, or callbacks from Web Services that have been called by the BPEL in an asynchronous way.

PartnerLinkType, Role, PortType

- **PartnerLinkType** defines the **conversational relationship** between two WSDL-defined partners by defining the **roles** played by each of the services **(invoker or client)** in the conversation and specifying the **portType (interface)** provided by each service to receive / transmit messages within the context of the conversation.
  - Each partnerLinkType has one or two roles (two roles in case of asynchronous callback), and each role has one portType
  - In the case of BPEL, one of the WSDL partners in the conversation will always be the central coordinator (i.e. the BPEL process!)
- **PartnerLinkType, roles and portType are defined in WSDL through the extensibility mechanism (out of scope)**
- **PortType has been renamed to Interface in BPEL2.0**
  - BPEL2.0 not widely adopted yet (out of scope)

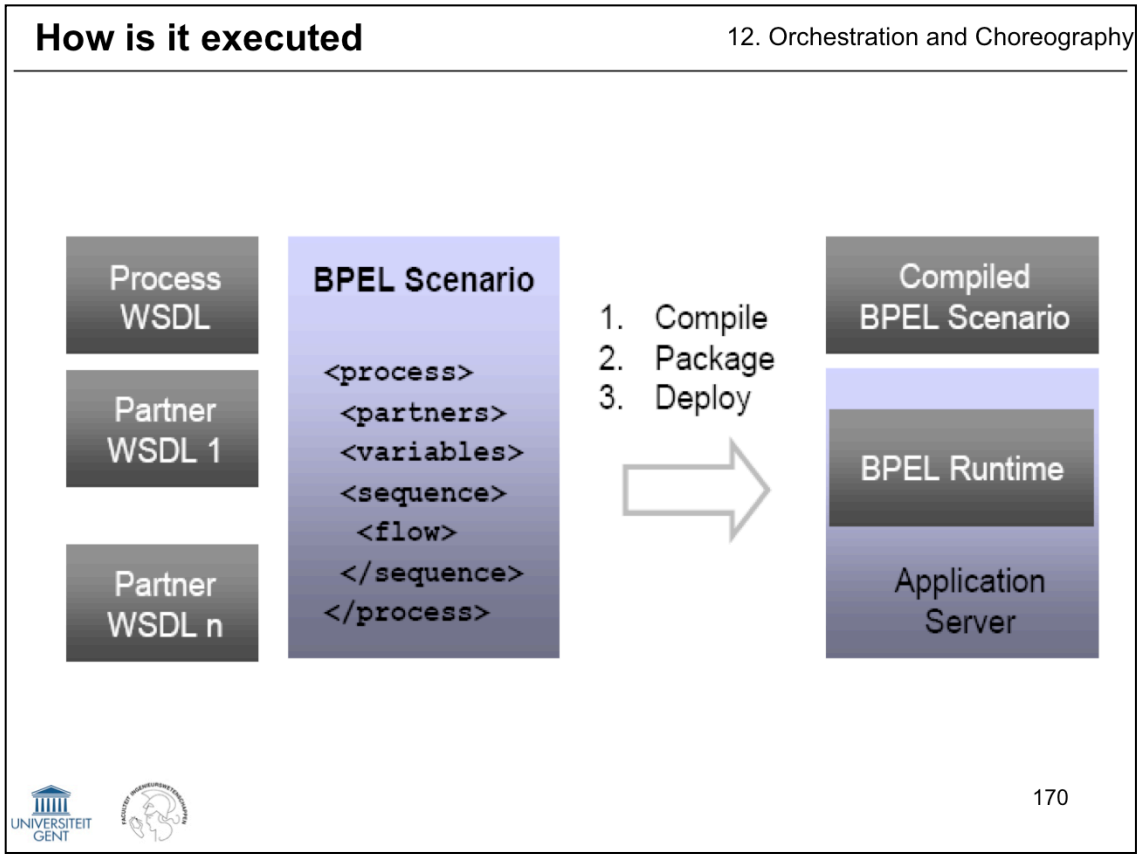
A partnerLinkType characterizes the conversational relationship between two services by defining the roles played by each of the services in the conversation and specifying the portType provided by each service to receive messages within the context of the conversation. Each <role> specifies exactly one WSDL portType.

Key BPEL concepts



Here we show an overview of these BPEL concepts and how they relate to each other. BPEL models a business process that performs actions on partner links. The actions on these partner links (web services) are defined by partner link types and roles (invoker or client), with each role defined by a port type (which defines the method, input message, output message to be used)

Actions and partner links are defined in the BPEL, while partner link types, roles and port types are defined in the WSDL.



Now that we have seen some of the major features of BPEL, we show how this is executed: On the lefthand side we have the process WSDL itself and the various partner WSDLs. The BPEL scenario then describes the actual bussiness process workflow.

These are then compiled, packaged and deployed on an application server, on which the BPEL runtime takes care of executing the compiled BPEL scenario.

Orchestration

BPEL: business process execution language

First version developed in 2002

Standardized through OASIS (Organisation for the Advancement of Structured Information Standards)

Used to standardize enterprise application integration

Also support in Netbeans

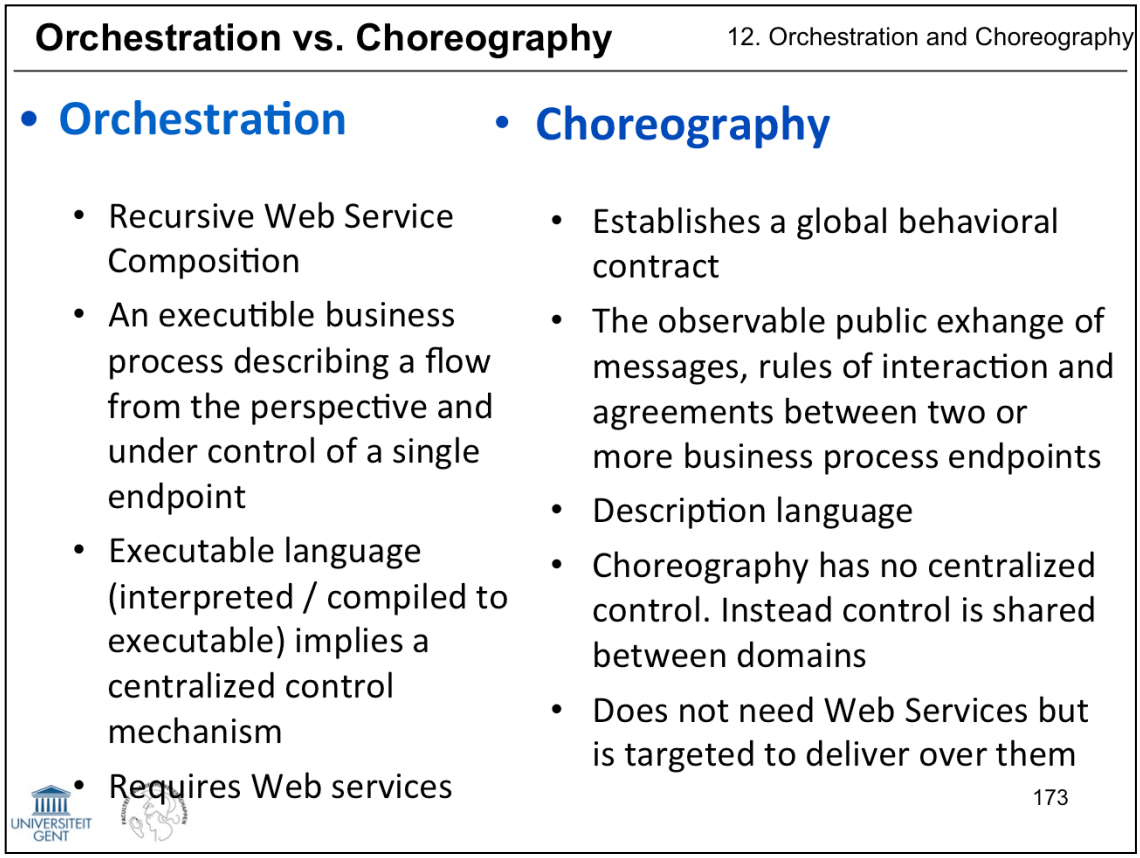
WS-Choreography

- **Working group in W3C, that is tasked with defining a language for describing peer to peer interactions of services from a neutral perspective**
- **Based on a formalized description of external observable behavior across domains**
- **Current status**
  - Requirements document (published March 2004)
  - Model Overview document (published April 2004),
  - 1st Working Draft of the WS-CDL specification (published April 2004)
  - Last Call Working Draft published Dec 2004.
  - Candidate Recommendation Sep 2005.

Now that we have discussed orchestration (i.e. business process started and executed by one controlling party), we will have a look at web service choreographies.

The W3C has a working group that defines a language for describing peer to peer interactions of services when there is no central coördinator of the business workflow. A web service choreography tries to formally describe the external observable behavior, as it is exactly this overall observable behavior that defines the business process.

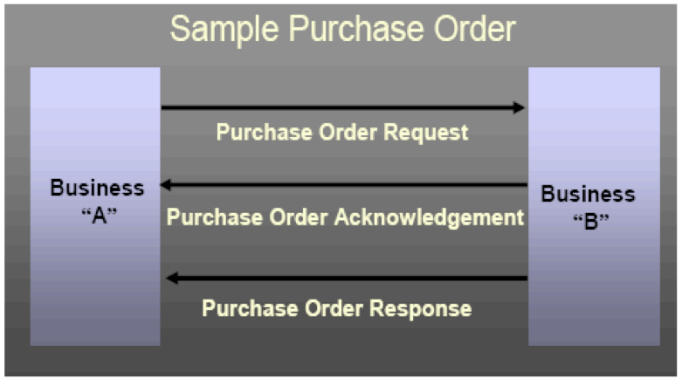
Currently the WS-CDL (Web Service – Choreography Description Language) specification is not yet a standard, but is still in the candidate recommendation state.



As a rehash of what we have seen regarding orchestration versus choreography, an orchestration is an executible business process under the control of a single endpoint. An orchestration is defined in an executable language (BPEL) that gets compiled and can be run by a BPEL runtime on an application server. Furthermore, orchestrations require the use of web services to implement endpoint activities.

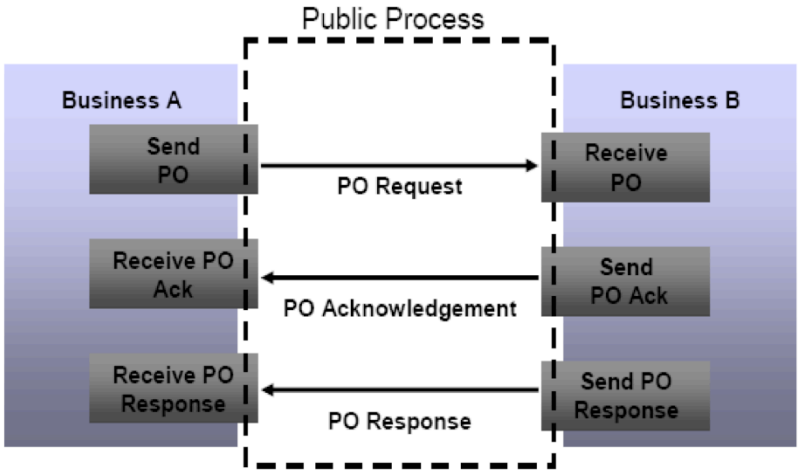
A choreography on the other hand establishes a global behavioral contract. This contract specifies (in a description language: WS-CDL) the observable public exchange of messages, rules of interaction and agreements between two or more business endpoints in the choreography. Choreographies have no centralized control (otherwise it would be an orchestration) and do not require web services but are targeted to work with them.

Sample Business Process



To give an example of a possible choreography we look at a purchase process between a business A and a business B. Business A initiates a purchase order, after which business B acknowledges that it has received the request. After processing the request Business B then passes a purchase order response message to business A.

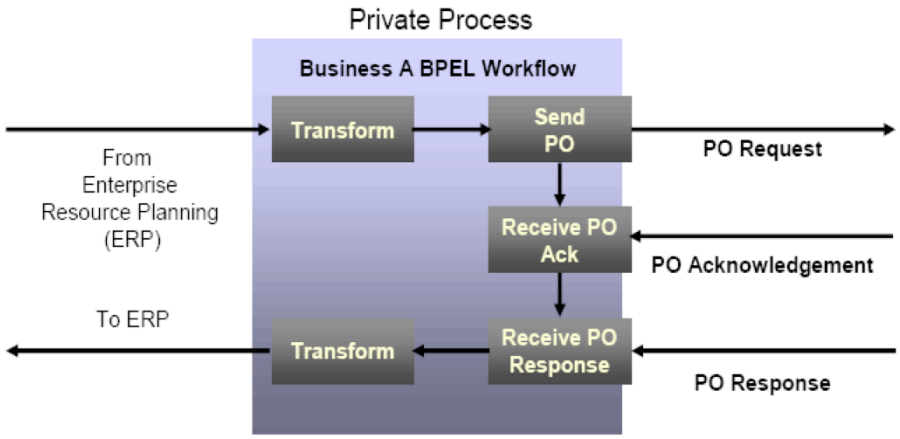
From a choreography perspective



Choreography – The observable public exchange of messages

If we look at this from a choreography perspective, the choreography is interested in the public observable behavior, i.e. the public exhange of messages between these two participants that leads to a global ‘state’

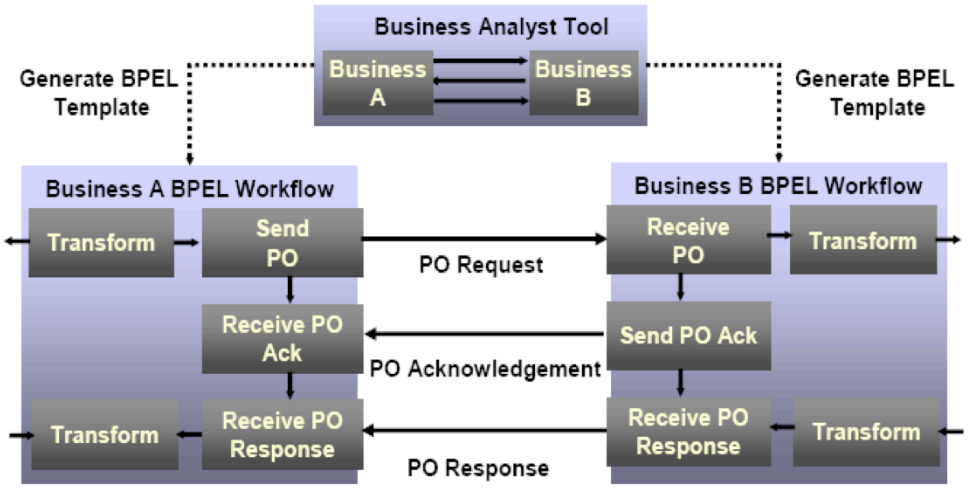
From an orchestration perspective



Orchestration – A private executable business process

Both of the businesses in this example are internally modelled as an orchestration, in that they both execute a private business process. Note that this can still mean that the the internals of business A a BPEL workflow is executed relying on external web services, but Business A has centralized control over all processes in this workflow.

Orchestration and choreography together



Two BPEL workflow templates reflecting a business agreement

If we now look at the global picture of our example again, we see that the top layer reflects the actual business agreement between the two participants in the choreography. This business agreement is modelled as a choreography by a business analyst tool, which in turn can generate BPEL templates for both Business A and Business B. Business A and B can implement these BPEL templates (as they see fit and without having to consult each other), and, if done correctly, they can execute the choreography (i.e. if both parties adhere to the BPEL template generated by the choreography / business analyst tool).

Conclusion

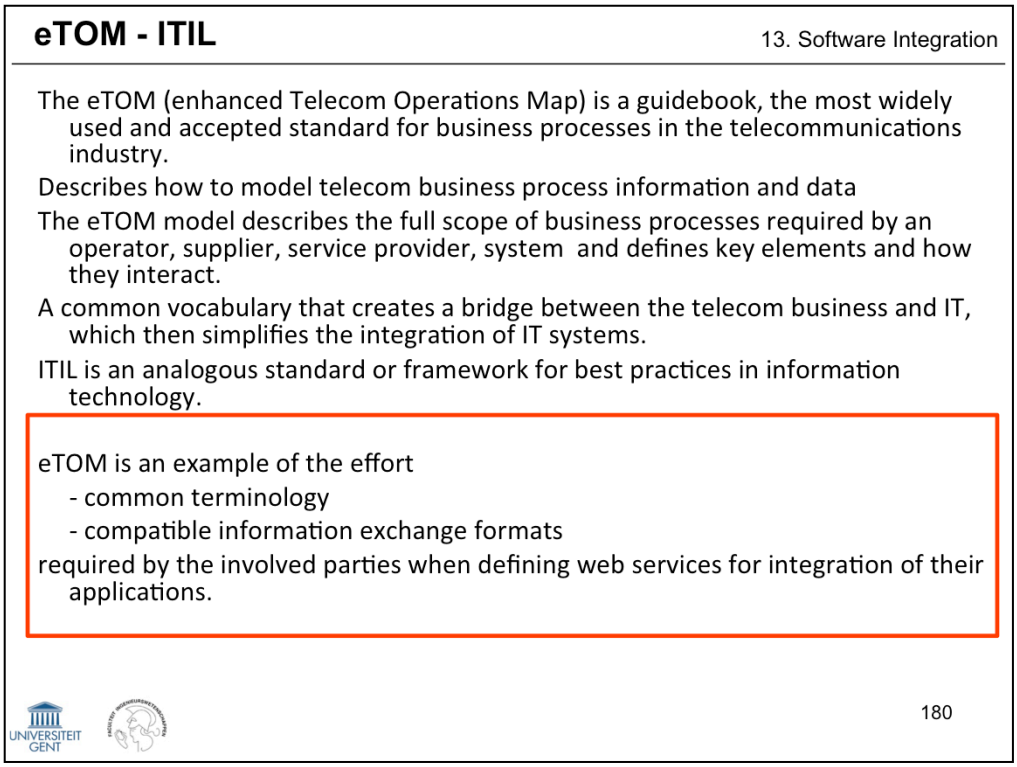
- **Orchestration and choreography are used to specify business processes**
- **What is the benefit?**
  - XML specification guarantees platform independence
  - Reuse of existing functionality
  - Offers high-level business process view / more distinction between business process and actual implementation
  - Distributed deployment
- **What is the problem**
  - Product incompatibilities
    - Improving over time
  - Lack of choreography tooling
    - Construction of WS-CDL and automated extraction of different partner's BPEL templates is still in research

To conclude we can say that orchestration and choreography are used to specify complex business processes. The benefit of these technologies is that due to the fact that both are specified in XML they can be platform independent. Both give a high-level view of the business process, allowing business flow designers to focus more on the business flow and less on the service implementation details.

The problem with orchestration and choreographies is that in reality a lot of products are proving to be incompatible (i.e. Web Services that cannot communicate with each other due to being developed by different tools). Web service choreographies also suffer from a lack of good tooling i.e. atm a designer cannot automatically extract BPEL templates from a WS-CDL definition.

Outline

9. Introduction : Service Oriented Architectures
10. Technology building blocks
    - SOAP
    - WSDL
    - WS-* standards
11. Java Web Services
12. Web Service Orchestration and Choreography
13. Software Integration
    - eTOM, ITIL



A summary of eTOM is presented here. It describes how to model telecom business process information and data.

It is an example of the effort (common terminology and compatible information exchange formats) required by the involved parties when defining web services for integration of their applications.

ITIL (Information Technology Infrastructure Library) is an analogous initiative originating from IT industry, detailing IT business processes and common vocabulary.