Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

Putting Services into Practice

Massimo Mecella
mecella@dis.uniroma1.it

with slides from Damiano Pozzi and Valerio Colaianni

Outline (1)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

- Introduction to Web Service Technologies (3 hour)
  1. XML
  2. WSDL
  3. UDDI
  4. WS-BPEL (formerly BPEL4WS)
  5. How to Concretely Develop a Web Service
     - AXIS 2
     - JBOSS

Outline (2)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

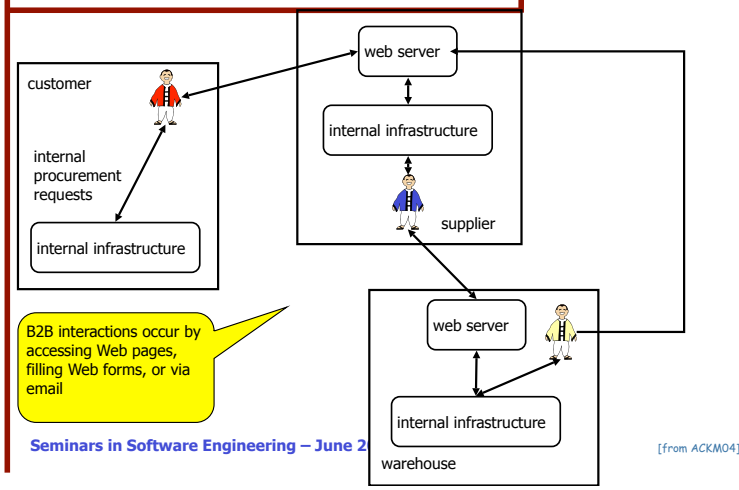
- Design and Use of a Lite Composition Tool (3 hours)
  1. Representing Service Behaviors in XML
  2. Practical Use
  3. Design Issues on Converting Composite Service Specifications into WS-BPEL and/or Java Orchestration Engine

Casati et al. – chapter 4 -- 8

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

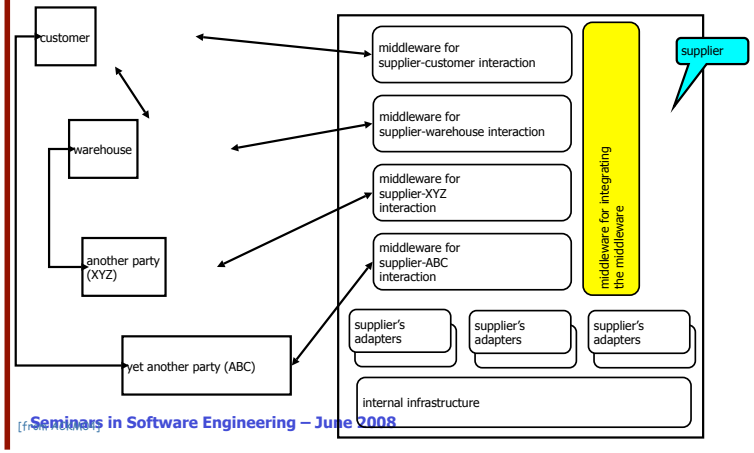
Introduction to Web Services Technologies

Motivations -- (naive) Business-to-Business Integration



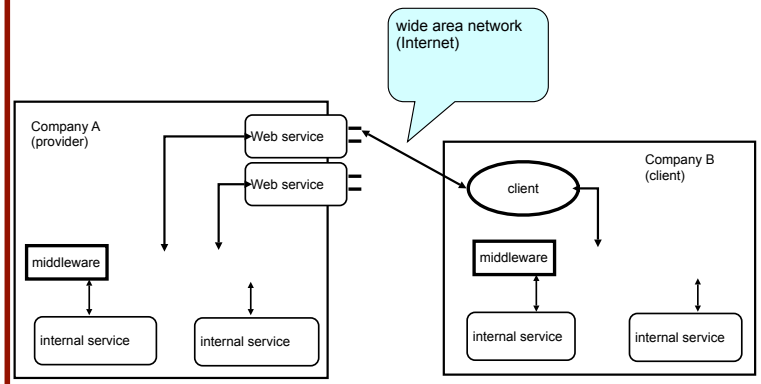
[from ACKM04]

Motivations -- WSs: the Evolution of Middleware and EAI Technologies (1)



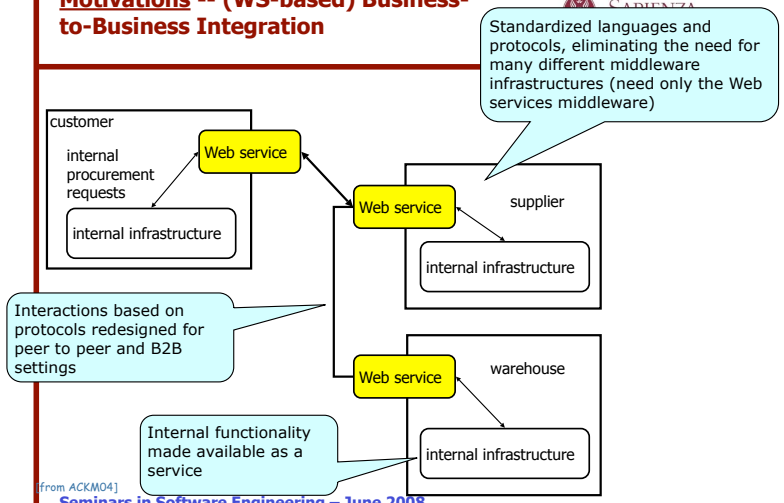
[from ACKM04]

Motivations -- WSs: the Evolution of Middleware and EAI Technologies (2)



[from ACKM04]

Motivations -- (WS-based) Business-to-Business Integration



[from ACKM04]

Motivations -- When Web Services Should Be Applied ?

- When it is no possible to easily manage deployment so that all requesters and providers are upgraded at once
- When components of the distributed system run on different platforms and vendor products
- When an existing application needs to be exposed over a network for use by unknown requesters

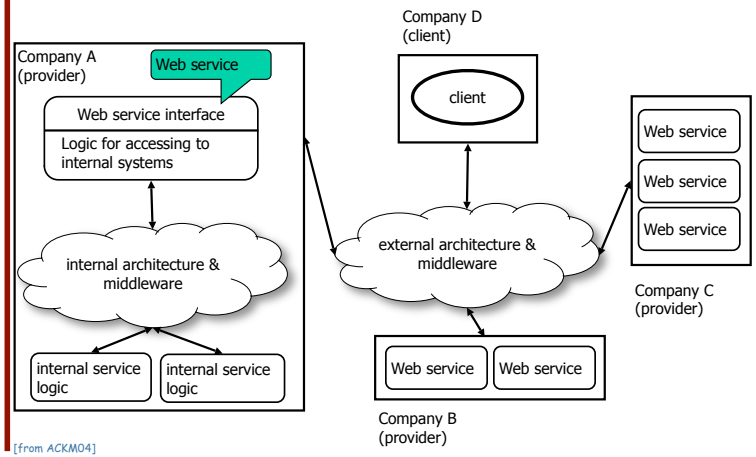
» *Web Services Architecture, W3C Working Group Note, 11 Feb. 2004,* `http://www.w3.org/TR/ws-arch/`

OVERVIEW

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

Two Architectures (and Middlewares)

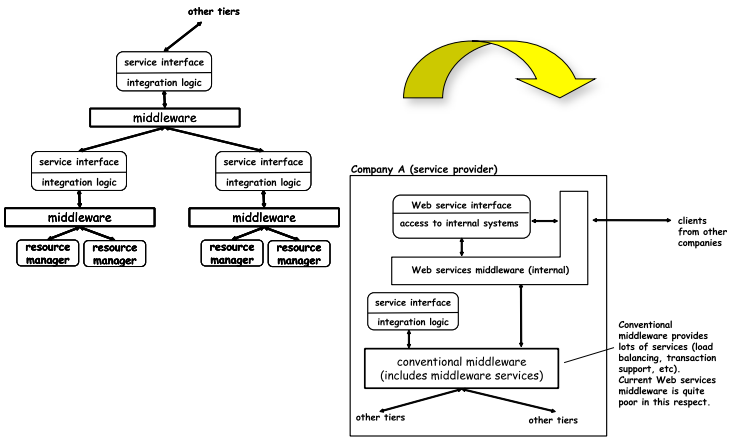
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma



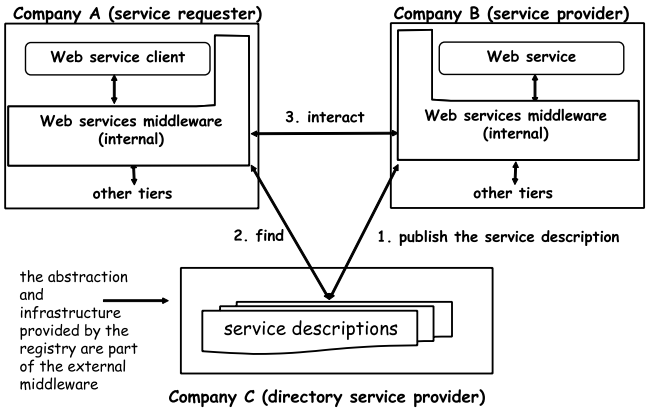
[from ACKM04]

The Internal Architecture

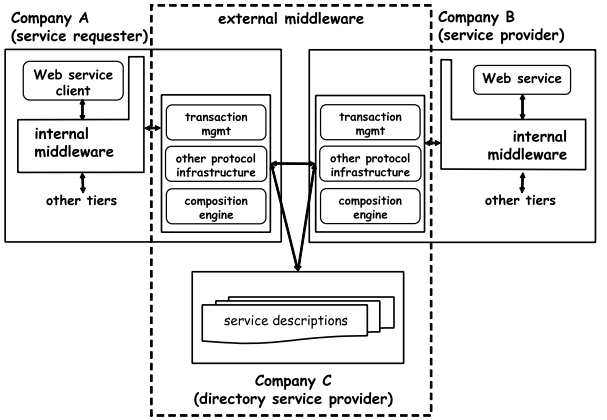
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma



The External Architecture



The External Middleware



XML BASICS

Extensible Markup Language

Extensible Markup Language (XML):

- Describes data objects called XML documents
- Is composed of markup language for structuring data
- Supports custom tags for definition, transmission, validation, and interpretation of data
- Conforms to Standard Generalized Markup Language (SGML)
- Has become a standard way to

A Simple XML Page: Example

```
<?xml version="1.0"?>
<employees>
  <employee>
    <employee_id>120</employee_id>
    <last_name>Weiss</last_name>
    <salary>8000</salary>
  </employee>
  <employee>
    <employee_id>121</employee_id>
    <last_name>Fripp</last_name>
    <salary>8200</salary>
  </employee>
</employees>
```


XML Document Structure

An XML document contains the following parts:

1. Prologue
2. Root element
3. Epilogue

```
<?xml version="1.0" encoding="WINDOWS-1252"?>   ①
<!-- this is a comment -->
<employees>                                      ②
 ...
</employees>

<?gifPlayer size="100,300" ?>                    ③
```


The XML Declaration

XML documents must start with an XML declaration.

The XML declaration:

- Looks like a processing instruction with the `xml` name. For example:

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<document-root>
...
</document-root>
```

- Must contain the `version` attribute
- May (optionally) include:
  - The `encoding` attribute
  - The `standalone` attribute
- Is optional in XML 1.0, but mandatory in XML 1.1

Components of an XML Document

XML documents comprise storage units containing:

- Parsed data, including the:
  - Markup (elements, attributes, and entities) used to describe the data they contain
  - Character data described by markup

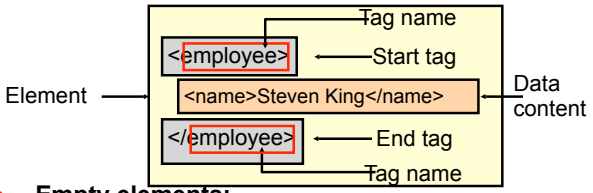
```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<employees>
  <employee id="100">
    <name>Rachael O&apos;Leary</name>
  </employee>
</employees>
```

- Unparsed data, such as textual or binary information (graphic and sound data), is left as entered.

```
<![CDATA[  ...unparsed data... ]]>
```

XML Elements

- An XML element has:
  - A start tag, end tag, and optional data content
  - Case-sensitive tags (start and end tags must match)



- Empty elements:
  - Do not contain any data
  - May appear as a single tag

```
<initials></initial>
<initials/>
```


Markup Rules for Elements

- There is one root element, sometimes called the top-level or document element.
- All elements:
  - Must have matching start and end tags, or be a self-closing tag (that is, an empty element)
  - Can contain nested elements such that their tags do not overlap
  - Have case-sensitive tag names subject to naming conventions (that is, they must start with a letter, contain no spaces, and not start with the letters `xml`)
  - May contain white space (spaces, tabs, new lines, and combinations of them) that is considered part of the element data content

XML Attributes

An XML attribute is a name-value pair that:

- Is specified in the start tag after the tag name

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<employees>
  <employee id="100" name='Rachael O&apos;Leary'>
    <salary>1000</salary>
  </employee>
</employees>
```

- Has a case-sensitive name
- Has a case-sensitive value that must be enclosed in matching single or double quotation marks
- Provides additional information about the XML document or XML elements

Using Elements Versus Attributes

```
<?xml version="1.0"?>
<employees>
  <employee>
    <id>100</id>
    <last_name>King</last_name>
    <salary>24000</salary>
  </employee>
</employees>
```

1 Elements

```
<?xml version="1.0"?>
<employees>
  <employee id="100" last_name="King"
            salary="24000">
    <job>President</job>
  </employee>
</employees>
```

2 Attributes

XML Entities

An XML entity:

- Is a unit of data storage
- Is identified by a case-sensitive name
- Is used as replacement text (substitute) when referencing its name between an ampersand (`&`), and a semicolon (`;`)

```
<comment>Salaries must not be &lt; 1000</comment>
```

- Has predefined names for special XML characters:
  - `<` for less than (<), and `>` for greater than (>)
  - `&` for ampersand (`&`)
  - `"` for double quotation mark (`"`)
  - `'` for single quotation mark (`'`)

XML Comments

XML comments:

- Start with `<!--`
- End with `-->`
- May appear anywhere in the character data of a document, and before the root element
- Are not elements, and can occupy multiple lines
- May not appear inside a tag or another comment

```
<?xml version="1.0" encoding="WINDOWS-1252"?>
<!-- Comment: This document has information about
     employees in the company -->
<employees>
 <name>Steven King</name> <!-- Full name -->
</employees>
```


A Well-Formed XML Document

Every XML document must be well-formed, such that:

- An XML document must have one root element
- An element must have matching start and end tag names, unless they are empty elements
- Elements can be nested, but cannot overlap
- All attribute values must be quoted
- Attribute names must be unique in the start tag of an element
- Comments and processing instructions do not appear inside tags
- The < or & special characters cannot appear in the character data of an element or attribute value

Comparing XML and HTML

- XML
  - Is a markup language for describing data
  - Contains user-defined markup elements
  - Is extensible
  - Is displayed as a document list in a Web browser
  - Conforms to rules for a well-formed document
- HTML
  - Is a markup language for formatting data in a Web browser
  - Contains predefined markup tags
  - Is not extensible
  - Does not conform to well-formed document rules

XML Development

XML documents can be developed by using:

- A simple text editor, such as Notepad
- A specialized XML Editor, such as XMLSpy

What Is a Document Type Definition?

A document type definition (DTD):

- Is the grammar for an XML document
- Contains the definitions of
  - Elements
  - Attributes
  - Entities
  - Notations
- Contains specific instructions that the XML parser interprets to check the document validity
- May be stored in a separate file (external)
- May be included within the document (internal)

Why Validate an XML Document

- Well-formed documents satisfy XML syntax rules, and not the business requirements of content and structure.
- Business rules often require validation of the content and structure of a document.
- XML documents must satisfy structural requirements imposed by the business model.
- A valid XML document can be reliably processed by XML applications.
- Validations can be performed by using a DTD or an

General DTD Rules

A DTD:

- Must provide a declaration for items used in an XML document, such as:
  - Elements
  - Attributes
  - Entities
- Is case-sensitive, but spacing and indentation are not significant
- May use XML comment syntax for documentation, but comments cannot appear inside declarations

The Contents of a DTD

A DTD contains declarations (that use the syntax shown) for:

- Elements:

```
<!ELEMENT element-name content-model>
```

- Attributes:

```
<!ATTLIST element-name attrib-name type default>
```

- Entities:

```
<!ENTITY entity-name "replacement text">
```

- Notations:

```
<!NOTATION notation_name SYSTEM "text">
```


Simple DTD Declaration: Example

Example of a simple DTD with element declarations:

```
<!ELEMENT employees (employee)>
<!ELEMENT employee (name)>
<!ELEMENT name (#PCDATA)>
```

A valid XML document based on the DTD:

```
<?xml version="1.0"?>
<employees>
  <employee>
    <name>Steven King</name>
  </employee>
</employees>
```

Note: All child elements must be defined.

Referencing the DTD

The XML document references the DTD:

- After the XML declaration and before the root, by using:

```
<!DOCTYPE employees [ ... ]>
```

- Externally with the `SYSTEM` or `PUBLIC` keywords:

```
<!DOCTYPE employees SYSTEM "employees.dtd">
```

```
<!DOCTYPE employees PUBLIC "-//formal-public-ID">
```

- Internally in the `<!DOCTYPE root [...]>` entry:

```
<?xml version="1.0"?>
<!DOCTYPE employees [
  <!ELEMENT employees (#PCDATA)>
]>
<employees>Employee Data</employees>
```

Note: Use the root element name after `<!DOCTYPE`.

Element Declarations

- Element declaration syntax:

```
<!ELEMENT element-name content-model>
```

- Four kinds of content models:

```
<!ELEMENT job EMPTY> <!-- Empty -->
```
①

```
<!-- Elements: single, ordered list, or choice -->
<!ELEMENT employees (employee)>
<!ELEMENT employee (employee_id,last_name,job_id)>
<!ELEMENT job_id (manager | worker)>
```
②

```
<!-- Mixed -->
<!ELEMENT last_name (#PCDATA)>
<!ELEMENT hire_date (date| (day,month,year))>
```
③

```
<!ELEMENT employee_id ANY><!-- Any -->
```
④

Attribute Declarations

- The syntax for declaring an attribute is:

```
<!ATTLIST element-name attrib-name type default>
```

- Attribute declaration requires:
  - An element name
  - An attribute name
  - An attribute type, specified as:
    CDATA, enumerated, ENTITY, ENTITIES, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, and NOTATION
  - An attribute default, specified as:
    #IMPLIED, #REQUIRED, #FIXED, or a literal value
- Example:

```
<!ELEMENT employee (employee_id, last_name)>
<!ATTLIST employee manager_id CDATA #IMPLIED>
```


CDATA and Enumerated Attribute Types

- CDATA: For character data values

```
<!ELEMENT employee (employee_id, last_name)>
<!ATTLIST employee manager_id CDATA #IMPLIED>
```

```
<employee manager_id="102">          <!-- XML -->
  <employee_id>104</employee_id>
  <last_name>Ernst</last_name>
</employee>
```

- Enumerated: For a choice from a list of values

```
<!ELEMENT employee (employee_id, last_name)>
<!ATTLIST employee gender (male|female) #IMPLIED>
```

```
<employee gender="male">             <!-- XML -->
  <employee_id>104</employee_id>
  <last_name>Ernst</last_name>
</employee>
```


NOTATION Declaration and Attribute Type

- Declaring a NOTATION:

```
<!NOTATION notation_name SYSTEM "text">
```

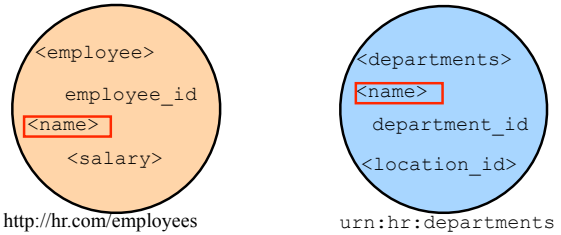
- The NOTATION attribute type represents a name of a NOTATION declared in the DTD:

```
<?xml version="1.0"?>
<!DOCTYPE photos [
<!ELEMENT photos (image+)>
<!ELEMENT image EMPTY>
<!NOTATION gif SYSTEM "image/gif">
<!NOTATION jpeg SYSTEM "image/jpeg">
<!ATTLIST image
   source CDATA #REQUIRED
   type NOTATION (gif | jpeg) #REQUIRED>
]>
<photos>
  <image source="myphoto.gif" type="gif"/>
  <image source="mypet.jpg" type="jpeg"/>
</photos>
```


What Is an XML Namespace?

An XML namespace:

- Is identified by a case-sensitive Internationalized Resource Identifier (IRI) reference (URL or URN)
- Provides universally unique names for a collection of names (elements and attributes)



Declaring XML Namespaces

Declare an XML namespace:

- With the `xmlns` attribute in an element start tag:
  - Assigned an IRI (URL, URI, or URN) string value
  - Provided with an optional namespace prefix
- With a namespace prefix after `xmlns:` to form qualified element names:

```
<dept:department
  xmlns:dept="urn:hr:department-ns">
...
</dept:department>
```

- Without a prefix to form a "default namespace":

```
<department xmlns="http://www.hr.com/departments">
...
</department>
```


XML Namespace Prefixes

A namespace prefix:

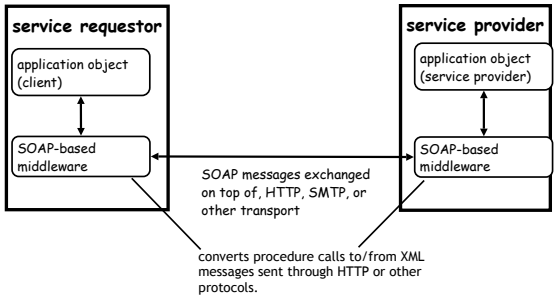
- May contain any XML character except a colon
- Can be declared multiple times as attributes of a single element, each with different names whose values can be the same or a different string
- Can be overridden in a child element by setting the value to a different string. For example:

```
<?xml version="1.0"?>
<emp:employee xmlns:emp="urn:hr:employee-ns">
  <emp:last_name>King</emp:last_name>
  <emp:address xmlns:emp="urn:hr:address-ns">
  500 Oracle Parkway
  </emp:address>
</emp:employee>
```

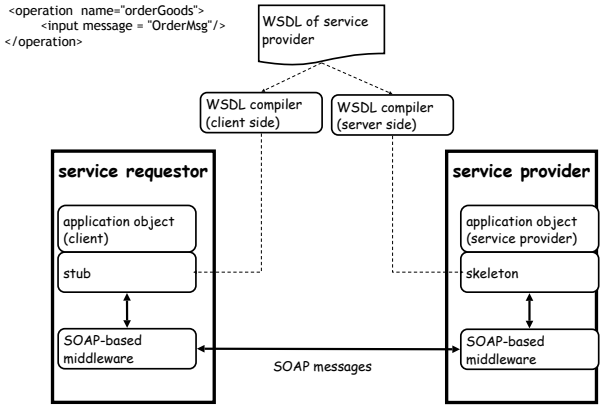

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

WEB SERVICE INFRASTRUCTURE

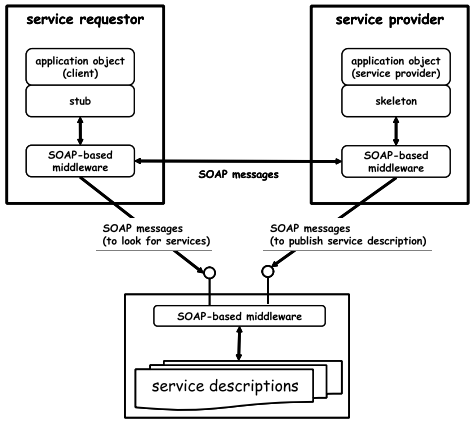
A Minimalist Infrastructure for Web Service



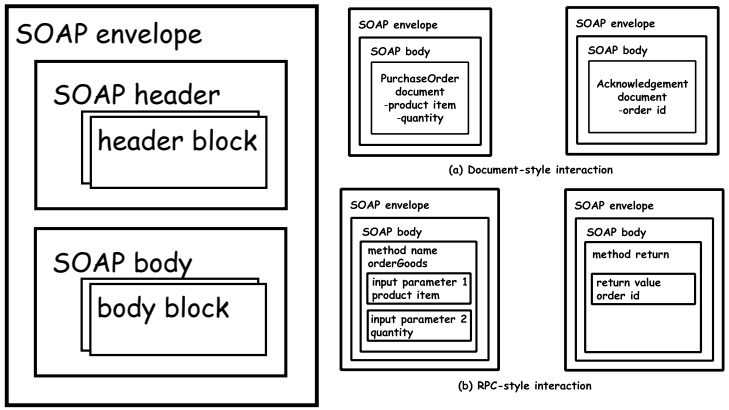
From Interfaces to Stub/Skeleton



Registry

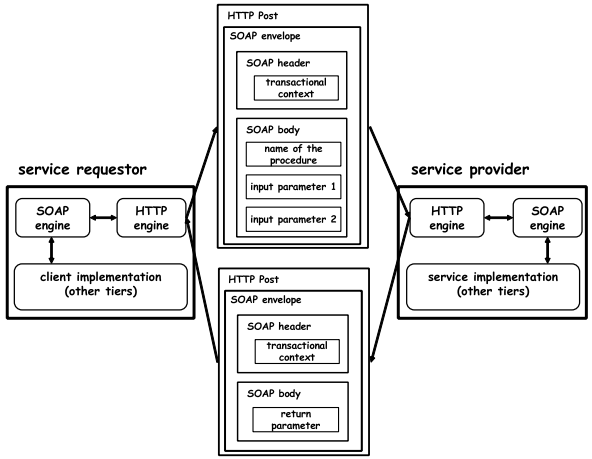


SOAP (1)

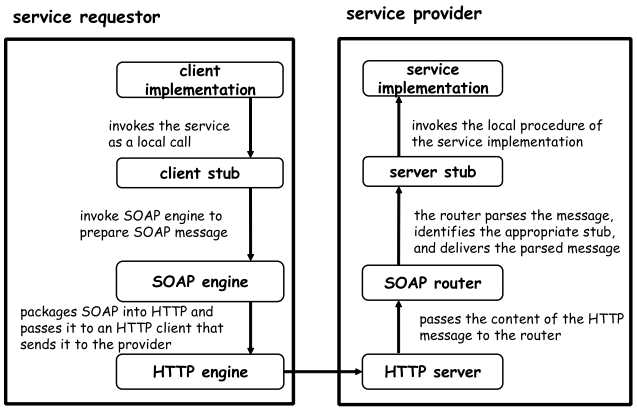


SOAP (2)

RPC with SOAP

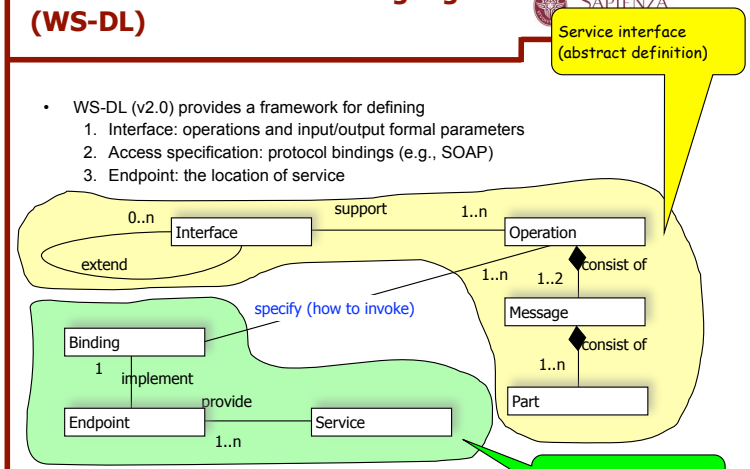


The Simplest SOAP Middleware

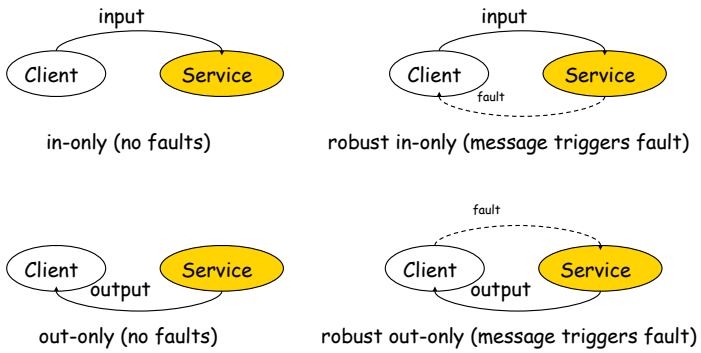


Web Service Definition Language (WS-DL)

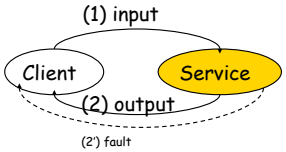
- WS-DL (v2.0) provides a framework for defining
  1. Interface: operations and input/output formal parameters
  2. Access specification: protocol bindings (e.g., SOAP)
  3. Endpoint: the location of service



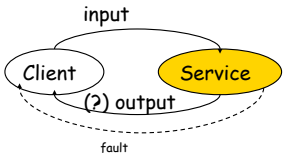
Message Exchange Patterns (1)



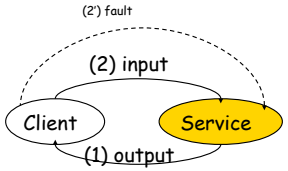
Message Exchange Patterns (2)



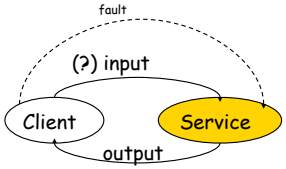
in-out (fault replaces message)



in-optional-out
(message triggers fault)

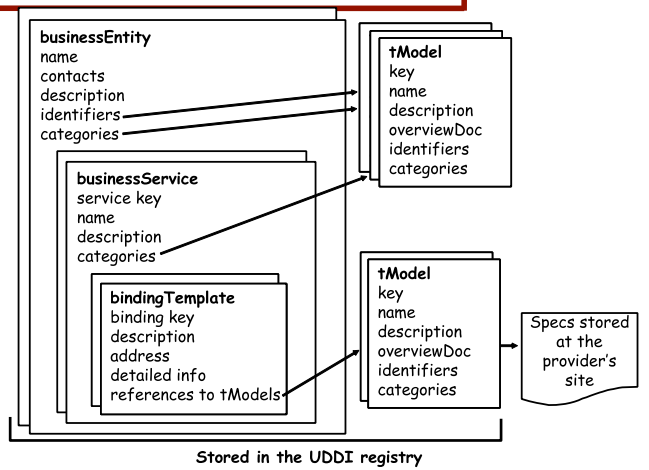


out-in (fault replaces message) une 2008



out-optional-in
(message triggers fault)

UDDI Data Structures



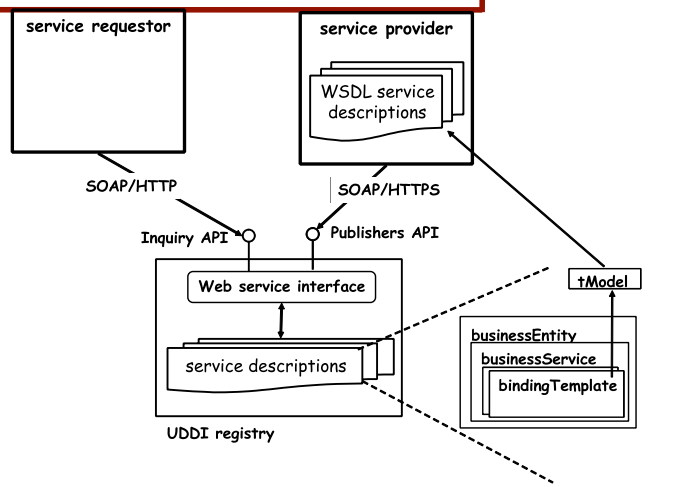
A Registry Not a Repository

overviewDoc (refer to WSDL specs and to API specs)

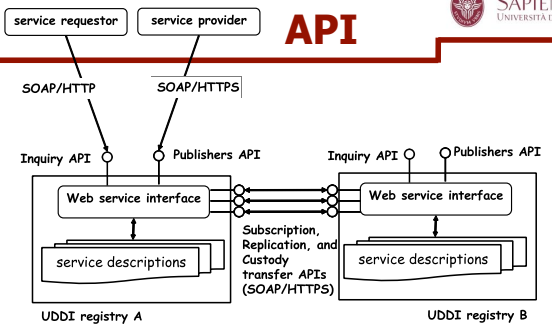
classification information (specifies that this tModel is about XML, WSDL, and SOAP specs)

```
<tModel tModelKey="uddi:uddi.org:v3_publication">
  <name>uddi-org:publication_v3</name>
  <description>UDDI Publication API V3.0</description>
  <overviewDoc>
     <overviewURL useType="wsdlInterface">
http://uddi.org/wsdl/uddi_api_v3_binding.wsdl#UDDI_Publication_SoapBinding
     </overviewURL>
  </overviewDoc>
  <overviewDoc>
     <overviewURL useType="text">
        http://uddi.org/pubs/uddi_v3.htm#PubV3
     </overviewURL>
  </overviewDoc>
  <categoryBag>
     <keyedReference keyName="uddi-org:types:wsdl"
        keyValue="wsdlSpec"
        tModelKey="uddi:uddi.org:categorization:types"/>
     <keyedReference keyName="uddi-org:types:soap"
        keyValue="soapSpec"
        tModelKey="uddi:uddi.org:categorization:types"/>
     <keyedReference keyName="uddi-org:types:xml"
        keyValue="xmlSpec"
        tModelKey="uddi:uddi.org:categorization:types"/>
     <keyedReference keyName="uddi-org:types:specification"
        keyValue="specification"
        tModelKey="uddi:uddi.org:categorization:types"/>
  </categoryBag>
</tModel>
```

UDDI and WSDL

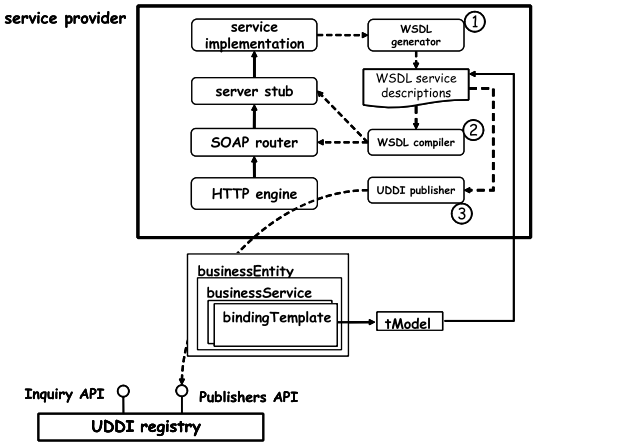


UDDI API

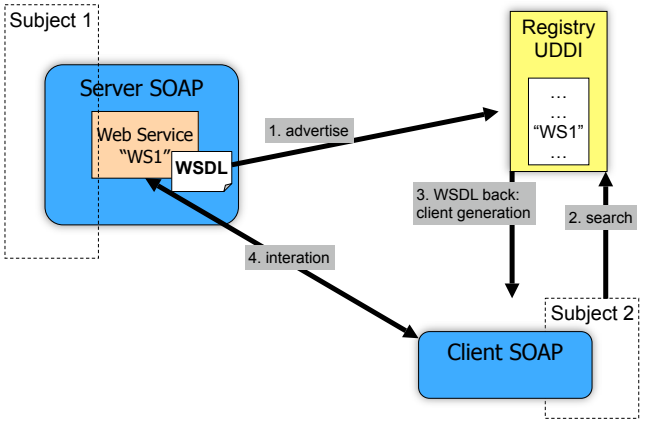



```
<?xml version="1.0"?>
<find_tModel generic="1.0" xmlns="urn:uddi-org:api">
  <categoryBag>
    <keyedReference tModelKey="UUID:C25893AF-1977-3528-36B5-4192C2AB9E2C"
                    keyName="uddi-org:types" keyValue="wsdlSpec"/>
    <keyedReference tModelKey="UUID:A15019C5-AE14-236C-331C-650857AE0221"
                    keyName="book pricing"
                    keyValue="36611349"/>
  </categoryBag>
```

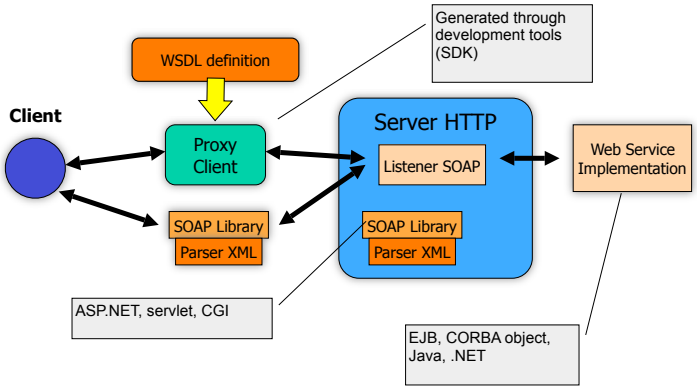
Putting All Together



Advertising and Using



Overall View



JUG Sardegna – http://www.jugsardegna.org

http://www.jugsardegna.org/vqwiki/jsp/Wiki?action=action_view_attachment&attachment=ArticoloAxis1PerJUG.pdf

http://www.jugsardegna.org/vqwiki/jsp/Wiki?action=action_view_attachment&attachment=ArticoloAxis2PerJUG.pdf

http://www.jugsardegna.org/vqwiki/jsp/Wiki?action=action_view_attachment&attachment=ArticoloAxis3PerJUG.pdf

Dispensa del corso di Progettazione del Software 2, nuova versione (da pubblicare a Luglio 2008)

CONCRETE DEVELOPMENT OF A WEB SERVICE

AXIS Development

- Axis/Axis2 is an open-source Web Service development and runtime environment designed as a plug-in of common Web container
  (e.g., Apache Tomcat)
  1. http://ws.apache.org/axis/ and http://ws.apache.org/axis2/
  2. webapps/axis to be installed in the servlet engine
- Basically is a servlet playing the role of SOAP Library and SOAP Listener

A Simple Service (1)

```
package miopackage;

public class SalutoWS {
   public String saluto(String name) {
   return "Ciao " + name + "!";
   }
}
```

- The class should be put in the right Axis directory, e.g., <TOMCAT_HOME>\webapps\axis\WEB-INF\classes\miopackage\SalutoWS.class
- Then the container should be made aware by deploying the class as a service
  1. Deployment descriptor <file>.wsdd

A Simple Service (2)

```
<deployment
xmlns="http://xml.apache.org/axis/wsdd/"
xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
   <service name="urn:SalutoWS" provider="java:RPC">
            <parameter name="className" value="miopackage.SalutoWS"/
   >
            <parameter name="allowedMethods" value="saluto"/>
            <parameter name="scope" value="Request"/>
   </service>
</deployment>
```

Life cycle of the service - transient (other values Application and Session)

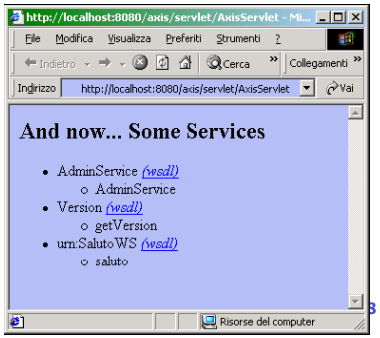
A Simple Service (3)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

Deployment through a specific tool

- `java org.apache.axis.client.AdminClient <file>.wsdd`

Development of a client



A Simple Service (3)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

Deploym

- java

Develop

```
import java.net.*;
import java.rmi.*;

import javax.xml.namespace.*;
import javax.xml.rpc.*;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;

public class ClientSalutoWS {
  public static void main(String[] args) {
    String messaggio = "";
    try {
      Call call = (Call) new Service().createCall();
      call.setTargetEndpointAddress(new
            URL("http://localhost:8080/axis/services/"));
      call.setOperationName(new QName("urn:SalutoWS", "saluto"));
      Object rispostaWS = call.invoke(new Object[]{"Nicola"});
      messaggio = "il Web service ha risposto: "+(String) rispostaWS;
    }
    catch (MalformedURLException ex) {
      messaggio = "errore: l'url non è esatta";
    }catch (ServiceException ex) {
      messaggio = "errore: la creazione della chiamata è fallita";
    }catch (RemoteException ex) {
      messaggio = "errore: l'invocazione del WS è fallita";
    }finally{
      System.out.println(messaggio);
    }
  }
}
```

A Simple Service (4)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

```
POST /axis/services/ HTTP/1.0
Content-Type: text/xml; charset=utf-8
Accept: application/soap+xml, application/dime, multipart/related, text/*
User-Agent: Axis/1.1
Host: 127.0.0.1
Cache-Control: no-cache
Pragma: no-cache
SOAPAction: ""
Content-Length: 442

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv=http://schemas.xmlsoap.org/soap/envelope/
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>

    <ns1:saluto soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        xmlns:ns1="urn:SalutoWS">
      <ns1:arg0 xsi:type="xsd:string">Nicola</ns1:arg0>
    </ns1:saluto>
  </soapenv:Body>
</soapenv:Envelope>
```

A Simple Service (4)

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Connection: close
Date: Fri, 21 Nov 2003 16:10:50 GMT
Server: Apache Tomcat/4.0.4-b2 (HTTP/1.1 Connector)

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd=http://www.w3.org/2001/XMLSchema
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:salutoResponse
        soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
        xmlns:ns1="urn:SalutoWS">
      <ns1:salutoReturn xsi:type="xsd:string">Ciao Nicola!</ns1:salutoReturn>
    </ns1:salutoResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

A More Complex Service (1) -- Serialization and (de)serialization


```
package miopackage.vo;

import java.io.*;

public class ProdottoVO implements Serializable {

  private int codice;
  private String nome;
  private boolean disponibile;
  private float prezzo;

  public ProdottoVO() {
  }
  public int getCodice() {
    return codice;
  }
  public void setCodice(int codice) {
    this.codice = codice;
  }
  public void setNome(String nome) {
    this.nome = nome;
  }
  public String getNome() {
    return nome;
  }
  public void setDisponibile(boolean disponibile) {
    this.disponibile = disponibile;
  }
  public boolean isDisponibile() {
    return disponibile;
  }
  public void setPrezzo(float prezzo) {
    this.prezzo = prezzo;
  }
  public float getPrezzo() {
    return prezzo;
  }

}
```

Application of the pattern "Data Transfer Object"

A More Complex Service (2) -- Serialization and (de)serialization


```
package miopackage;

import java.util.*;
import miopackage.vo.*;

public class CatalogoWS {

  public CatalogoWS() { }

  public Collection getListaProdotti() {
    Vector lista = new Vector();

    //prodotto1
    ProdottoVO prodotto1 = new ProdottoVO();
    prodotto1.setCodice(1);
    prodotto1.setNome("Lettore MP3");
    prodotto1.setDisponibile(true);
    prodotto1.setPrezzo(25.99f);
    lista.add(prodotto1);

    //prodotto2
    ProdottoVO prodotto2 = new ProdottoVO();
    prodotto2.setCodice(2);
    prodotto2.setNome("Display LCD");
    prodotto2.setDisponibile(true);
    prodotto2.setPrezzo(199.99f);
    lista.add(prodotto2);

    return lista;
  }

  public ProdottoVO getProdotto(int codice) {
    ProdottoVO prodotto = new ProdottoVO();
    prodotto.setCodice(codice);
    prodotto.setNome("Sub Woofer attivo");
    prodotto.setDisponibile(true);
    prodotto.setPrezzo(154.99f);
    return prodotto;
  }
}
```

A More Complex Service (3) -- Serialization and (de)serialization


```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="urn:catalogoWS" provider="java:RPC" style="rpc" use="encoded">
    <parameter name="className" value="miopackage.CatalogoWS"/>
    <parameter name="allowedMethods" value="getProdotto, getListaProdotti"/>
    <parameter name="scope" value="Session"/>
    <beanMapping qname="myNS:ProdottoVO" xmlns:myNS="urn:catalogoWS"
                 languageSpecificType="java:miopackage.vo.ProdottoVO"/>
  </service>
</deployment>
```

Compact form for.
Use the general form if you want to personally manage the serialization step

```
<typeMapping
  xmlns:ns=" urn:catalogoWS "
  qname=" myNS:ProdottoVO "
  type="java:miopackage.vo.ProdottoVO "
  serializer="org.apache.axis.encoding.ser.BeanSerializerFactory"
  deserializer="org.apache.axis.encoding.ser.BeanDeserializerFactory"
  encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
/>
```

A More Complex Service (4) -- Serialization and (de)serialization


```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions    targetNamespace="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns="http://schemas.xmlsoap.org/wsdl/"     xmlns:apachesoap="http://xml.apache.org/xml-
soap"                    xmlns:impl="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns:intf="http://localhost:8080/axis/services/urn:catalogoWS"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"     xmlns:tns1="urn:catalogoWS"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <wsdl:types>
    <schema targetNamespace="urn:catalogoWS" xmlns="http://www.w3.org/2001/XMLSchema">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <complexType name="ProdottoVO">
          <sequence>
            <element name="codice" type="xsd:int"/>
            <element name="disponibile" type="xsd:boolean"/>
            <element name="nome" nillable="true" type="xsd:string"/>
            <element name="prezzo" type="xsd:float"/>
          </sequence>
        </complexType>
    </schema>
  </wsdl:types>
  <wsdl:message name="getListaProdottiRequest">
  </wsdl:message>
  <wsdl:message name="getProdottoResponse">
    <wsdl:part name="getProdottoReturn" type="tns1:ProdottoVO"/>
  </wsdl:message>
  <wsdl:message name="getListaProdottiResponse">
    <wsdl:part name="getListaProdottiReturn" type="soapenc:Array"/>
  </wsdl:message>
  <wsdl:message name="getProdottoRequest">
    <wsdl:part name="codice" type="xsd:int"/>
  </wsdl:message>
```

A More Complex Service (5) -- Serialization and (de)serialization


```
<wsdl:portType name="CatalogoWS">
  <wsdl:operation name="getListaProdotti">
    <wsdl:input message="impl:getListaProdottiRequest" name="getListaProdottiRequest"/>
    <wsdl:output message="impl:getListaProdottiResponse"
                 name="getListaProdottiResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getProdotto" parameterOrder="codice">
    <wsdl:input message="impl:getProdottoRequest" name="getProdottoRequest"/>
    <wsdl:output message="impl:getProdottoResponse" name="getProdottoResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="urn:catalogoWSSoapBinding" type="impl:CatalogoWS">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getListaProdotti">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getListaProdottiRequest">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://miopackage" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="getListaProdottiResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://localhost:8080/axis/services/urn:catalogoWS"
                 use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getProdotto">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getProdottoRequest">
```


A More Complex Service (6) -- Serialization and (de)serialization


```
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://miopackage" use="encoded"/>
    </wsdl:input>
    <wsdl:output name="getProdottoResponse">
      <wsdlsoap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
                 namespace="http://localhost:8080/axis/services/urn:catalogoWS"
                 use="encoded"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
<wsdl:service name="CatalogoWSService">
  <wsdl:port binding="impl:urn:catalogoWSSoapBinding" name="urn:catalogoWS">
    <wsdlsoap:address location="http://localhost:8080/axis/services/urn:catalogoWS"/>
  </wsdl:port>
</wsdl:service>
</wsdl:definitions>
```


A More Complex Service (7) -- Serialization and (de)serialization


```
package miopackage;

import java.net.*;
import java.rmi.*;
import java.util.*;

import javax.xml.namespace.*;
import javax.xml.rpc.*;

import miopackage.vo.*;
import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.ser.*;

public class ClientCatalogoWS {

  public static void main(String[] args) {
    try {
      Call call = (Call) new Service().createCall();
      call.setTargetEndpointAddress(new URL("http://localhost:8080/axis/services/"));

      QName qnameProd = new QName("urn:catalogoWS", "ProdottoVO");
      Class classeProd = ProdottoVO.class;
      call.registerTypeMapping(classeProd, qnameProd, BeanSerializerFactory.class,
                                                      BeanDeserializerFactory.class);

      //richiamo il metodo getProdotto
      call.setOperationName(new QName("urn:catalogoWS", "getProdotto"));
      Object rispostaWS = call.invoke(new Object[]{new Integer(1)});
      ProdottoVO prodotto = (ProdottoVO) rispostaWS;
      System.out.println("Il prodotto scelto è:");
      visualizza(prodotto);
```

Cfr. with the wsdd

A More Complex Service (8) -- Serialization and (de)serialization


```
  //richiamo il metodo getListaProdotti
  call.setOperationName(new QName("urn:catalogoWS", "getListaProdotti"));
  rispostaWS = call.invoke(new Object[]{});
  Collection lista = (Collection) rispostaWS;
  System.out.println("Il catalogo comprende:");
  Iterator iter = lista.iterator();
  while (iter.hasNext()) {
    ProdottoVO item = (ProdottoVO) iter.next();


      visualizza(item);
    }
  } catch (Exception ex) {
    System.out.println("Si è verificata l'eccezione: " + ex.toString());
  }
}

public static void visualizza(ProdottoVO prodotto) {
  if (prodotto == null) {
    return;
  }
  System.out.println("nome: " + prodotto.getNome());
  System.out.println("codice: " + prodotto.getCodice());
  String disponibile = (prodotto.isDisponibile()) ? "si" : "no";
  System.out.println("disponibile: " + disponibile);
  System.out.println("prezzo: " + prodotto.getPrezzo());
}
}
```

A More Complex Service (9) -- Serialization and (de)serialization

Richiesta

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getProdotto
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="urn:catalogoWS">
      <ns1:arg0 xsi:type="xsd:int">1</ns1:arg0>
    </ns1:getProdotto>
  </soapenv:Body>
</soapenv:Envelope>
```

Risposta

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getProdottoResponse
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="urn:catalogoWS">
      <ns1:getProdottoReturn href="#id0"/>
    </ns1:getProdottoResponse>
    <multiRef id="id0" soapenc:root="0"
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns2:ProdottoVO"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns2="urn:catalogoWS">
      <codice xsi:type="xsd:int">1</codice>
      <disponibile xsi:type="xsd:boolean">true</disponibile>
      <nome xsi:type="xsd:string">Sub Woofer attivo</nome>
      <prezzo xsi:type="xsd:float">154.99</prezzo>
    </multiRef>
  </soapenv:Body>
</soapenv:Envelope>
```


A More Complex Service (10) -- Serialization and (de)serialization

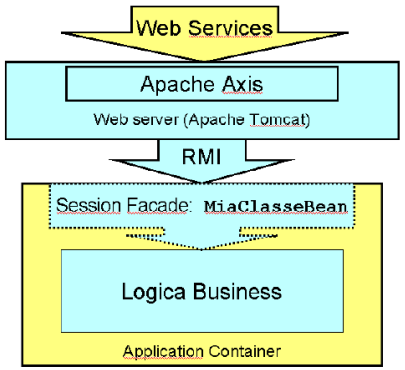
Richiesta

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getListaProdotti
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="urn:catalogoWS"/>
  </soapenv:Body>
</soapenv:Envelope>
```

Risposta

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ns1:getListaProdottiResponse
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns1="urn:catalogoWS">
      <ns1:getListaProdottiReturn href="#id0"/>
    </ns1:getListaProdottiResponse>
    <multiRef id="id0" soapenc:root="0"
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns2:Vector"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
        xmlns:ns2="http://xml.apache.org/xml-soap">
      <item href="#id1"/>
      <item href="#id2"/>
    </multiRef>
    <multiRef id="id1" soapenc:root="0"
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns3:ProdottoVO" xmlns:ns3="urn:catalogoWS"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <codice xsi:type="xsd:int">1</codice>
      <disponibile xsi:type="xsd:boolean">true</disponibile>
      <nome xsi:type="xsd:string">Lettore MP3</nome>
      <prezzo xsi:type="xsd:float">25.99</prezzo>
    </multiRef>
    <multiRef id="id2" soapenc:root="0"
        soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        xsi:type="ns4:ProdottoVO" xmlns:ns4="urn:catalogoWS"
        xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
      <codice xsi:type="xsd:int">2</codice>
      <disponibile xsi:type="xsd:boolean">true</disponibile>
      <nome xsi:type="xsd:string">Display LCD</nome>
      <prezzo xsi:type="xsd:float">199.99</prezzo>
    </multiRef>
  </soapenv:Body>
</soapenv:Envelope>
```


Services and Application Servers (1)



Services and Application Servers (2)

```
package miopackage;

import java.rmi.*;
import javax.ejb.*;
import miopackage.vo.* ;

public interface MiaClasse extends EJBObject{

  public RisultatoVO getRisultato(MioVO  vo);

}
```

```
package miopackage;

import java.rmi.*;
import javax.ejb.*;

public interface MiaClasseHome extends EJBHome {

  public MiaClasse create() throws RemoteException, CreateException;

}
```

Services and Application Servers (3)


```
<deployment
  xmlns="http://xml.apache.org/axis/wsdd/"
  xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">

  <service name="urn:nomeServizio" provider="java:EJB">
    <parameter name="jndiURL" value="jnp://nomehost:1099"/>
    <parameter name="jndiContextClass" value="org.jnp.interfaces.NamingContextFactory"/>
    <parameter name="beanJndiName" value="nomeJNDImiaEjb"/>
    <parameter name="homeInterfaceName" value="miopackage.MiaClasseHome"/>
    <parameter name="remoteInterfaceName" value="miopackage.MiaClasse"/>
    <parameter name="allowedMethods" value="getRisultato"/>
    <parameter name="scope" value="Session"/>
    <beanMapping qname="myNS1:MioVO" xmlns:myNS1=" urn:nomeServizio"
                 languageSpecificType="java:miopackage.vo.MioVO"/>
    <beanMapping qname="myNS2: RisultatoVO " xmlns:myNS2=" urn:nomeServizio"
                 languageSpecificType="java:miopackage.vo. RisultatoVO "/>

  </service>
</deployment>
```

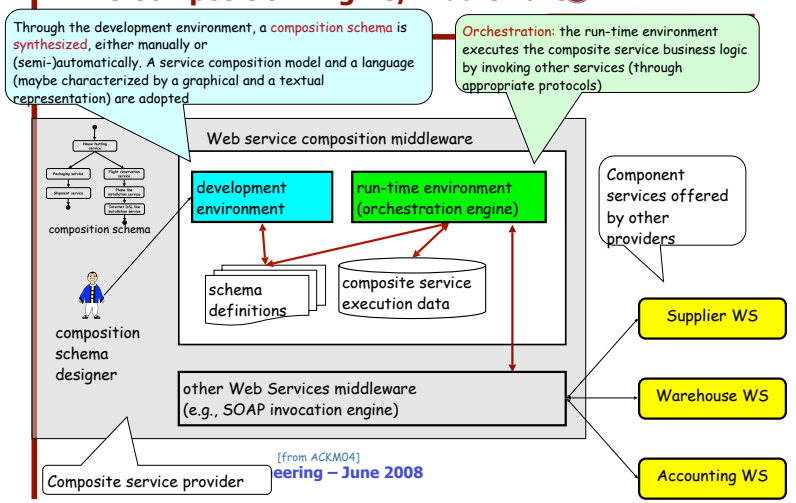
To deploy
- `MiaClasseHome`, `MiaClasse` and `RisultatoVO` into `WEB-INF\classes`
- `Jboss-client.jar`, `jboss-j2ee.jar`, etc. into `WEB-INF\lib`

COMPOSITION AND WS-BPEL

Composition

- Deals with the implementation of an application (in turn offered as a service) whose application logic involves the invocation of operations offered by other services
  1. The new service is the *composite* service
  2. The invoked services are the *component* services

The Composition Engine/Middleware



[from ACKM04]

Synthesis and Orchestration

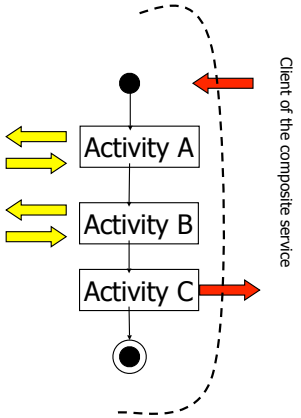
- (Composition) Synthesis: building the specification of the composite service (i.e., the composition schema)
  1. Manual
  2. Automatic
- Orchestration: the run-time management of the composite service (invoking other services, scheduling the different steps, etc.)
  1. Composition schema is the “program” to be executed
  2. Similarities with WfMSs (Workflow Management Systems)

Composition Schema

- A composition schema specifies the “process” of the composite service
  1. The “workflow” of the service
- Different clients, by interacting with the composite service, satisfy their specific needs (reach their goals)
  1. A specific execution of the composition schema for a given client is an orchestration instance

Business Process Execution Language for Web Services (WS-BPEL)

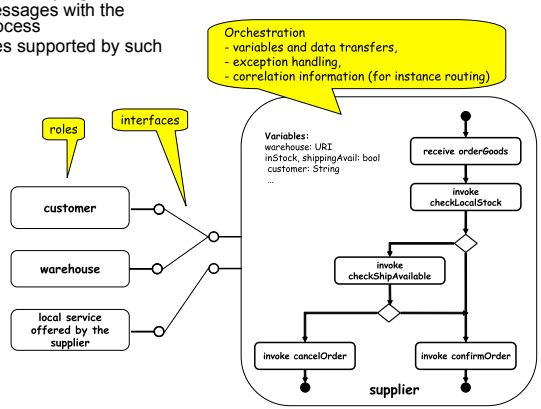
- Allows specification of composition schemas of Web Services
  1. Business processes as coordinated interactions of Web Services
  2. Business processes as Web Services
- Allows abstract and executable processes
- Influenced from
  1. Traditional flow models
  2. Structured programming
  3. Successor of WSFL and XLANG



WS-BPEL Specification

- An XML document specifying
- Roles exchanging messages with the composite service/process
- The (WSDL) interfaces supported by such roles

The orchestration of the process
- Variables and data transfer
- Exception handling
- Correlation information



Process Model
(Activities)

- Primitive
  1. `invoke`: to invoke a Web Service (in-out) operation
  2. `receive`: to wait for a message from an external source
  3. `reply`: to reply to an external source message
  4. `wait`: to remain idle for a given time period
  5. `assign`: to copy data from one variable to another
  6. `throw`: to raise exception errors
  7. `empty`: to do nothing
- Structured
  1. `sequence`: sequential order
  2. `switch`: conditional routing
  3. `while`: loop iteration
  4. `pick`: choices based on events
  5. `flow`: concurrent execution (synchronized by `link`s)
  6. `scope`: to group activities to be treated "transactionally" (managed by the same fault handler, within the same transactional context)

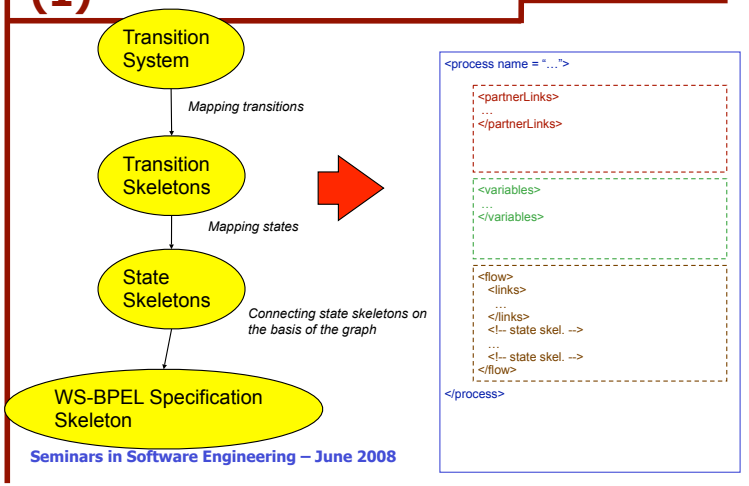
A link connects exactly one source activity S to exactly one target activity T; T starts only after S ends. An activity can have multiple incoming (possibly with join conditions) and outgoing links. Links can be guarded

Process Model
(Data Manipulation and Exception Handling)

- Blackboard approach
  1. a blackboard of variables is associated to each orchestration instance (i.e., a shared memory within an orchestration instance)
  2. variables are not initialized at the beginning; they are modified (read/write) by assignments and messages
  3. manipulation through XPath
- Try-catch-throw approach
  1. definition of fault handlers
  2. … but also event handlers and compensation handlers (for managing transactionality as in the SAGA model)

A (NICE) EXAMPLE

From a TS to WS-BPEL (1)



From a TS to WS-BPEL (2)

***Intuition* [Baina etal CAISE04, Berardi etal VLDB-TES04]**

1. Each transition corresponds to a WS-BPEL pattern consisting of *(i)* an **<onMessage>** operation (in order to wait for the input from the client of the composite service), *(ii)* followed by the effective logic of the transition, and then *(iii)* a final operation for returning the result to the client. Of course both before the effective logic and before returning the result, messages should be copied forth and back in appropriate variables
2. All the transitions originating from the same state are collected in a **<pick>** operation, having as many **<onMessage>** clauses as transitions originating from the state
3. The WS-BPEL file is built visiting all the nodes of the graph, starting from the initial state and applying the previous rules.

N.B.: (1) and (2) works for in-out interactions (the ones shown in the following). Simple modifications are needed for in-only, robust-in-only and in-optional-out. The other kinds of interactions implies a proactive behaviour of the composite service, possibly guarded by **<onAlarm>** blocks.

Transition Skeletons


```
<onMessage … >
    <sequence>
        <assign>
            <copy>
                <from variable="input" ... />
                <to variable="transitionData" ... />
            </copy>
        </assign>
        < !-- logic of the transition -->
        <assign>
            <copy>
                <from variable="transitionData" ... />
                <to variable="output" ... />
            </copy>
        </assign>
        <reply ... />
    </sequence>
</onMessage>
```


State Skeletons

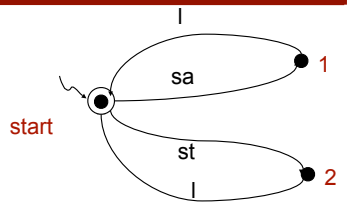
- N transitions from state S_i are mapped onto:

```
<pick name = "Si">
    <!-- transition #1 -->
    <onMessage … >
        <!-- transition skeleton -->
    </onMessage>
    … … …
    <!-- transition #N -->
    <onMessage … >
        <!-- transition skeleton -->
    </onMessage>
</pick>
```


Mapping the TS

- All the **<pick>** blocks are enclosed in a surrounding **<flow>**; the dependencies are modeled as **<link>**s
  1. **<link>**s are controlled by specific variables **s_i-to-s_j** that are set to TRUE iff the transition $S_i \rightarrow S_j$ is executed
  2. Each state skeleton has many outgoing **<link>**s as states connected in output, each going to the appropriate **<pick>** block
  3. Transitions going back into the initial state should not be considered, as they can be represented as the start of a new instance

An Example (1)




```
<partnerLinks>
  <!-- The "client" role represents the requester of this composite service -->
      <partnerLink name="client"
                   partnerLinkType="tns:Transition"
                   myRole="MP3ServiceTypeProvider"
                   partnerRole="MP3ServiceTypeRequester"/>
      <partnerLink name="service"
                   partnerLinkType="nws:MP3CompositeService"
                   myRole="MP3ServiceTypeRequester"
                   partnerRole="MP3ServiceTypeProvider"/>
</partnerLinks>
```


An Example (2)

```
<variables>
      <variable name="input" messageType="tns:listen_request"/>
      <variable name="output" messageType="tns:listen_response"/>
    <variable name="dataIn" messageType="nws:listen_request"/>
    <variable name="dataOut" messageType="nws:listen_response"/>
</variables>

    <pick>
      <onMessage partnerLink="client"
            portType="tns:MP3ServiceType"
            operation="listen"
            variable="input">
            <sequence>
                <assign>
                    <copy>
                      <from variable="input" part="selectedSong"/>
                      <to variable="dataIn" part="selectedSong"/>
                    </copy>
                </assign>
                ... ...
                <assign>
                    <copy>
                      <from variable="dataOut" part="MP3FileURL"/>
                      <to variable="output" part="MP3FileURL"/>
                    </copy>
                </assign>
                <reply name="replyOutput"
                     partnerLink="client"
                     portType="tns:MP3ServiceType"
                     operation="listen"
                     variable="output"/>
            </sequence>
      </onMessage>
    ... ...
    </pick>
```


An Example (3)

A new instance is created in the initial state. This resolve also the presence of the cycles without the need of enclosing <while>

The <sa> transition skeleton should set variables:
start-to-1 = TRUE
start-to-2 = FALSE

The <st> transition skeleton should set variables:
start-to-1 = FALSE
start-to-2 = TRUE

```
<process suppressJoinFailure = "no">
  <flow>
  <links>
     <link name="start-to-1"/>
     <link name="start-to-2"/>
   </links>

   <pick createInstance = "yes">
      <onMessage="sa">
         <sequence>
            <copy>...</copy>
            ... ...
            <copy>...</copy>
            <reply ... />

         </sequence>
      </onMessage>
      <onMessage="st">
         <sequence>
            <copy>...</copy>
            ... ...
            <copy>...</copy>
            <reply ... />
         </sequence>
      </onMessage>
      <source linkName="start-to-1" transitionCondition = "bpws:getVariableData('start-to-1') = 'TRUE' " />
      <source linkName="start-to-2" transitionCondition = "bpws:getVariableData('start-to-2') = 'TRUE' " />
   </pick>
```


An Example (4)

```
   <pick>
    <onMessage="I">
         <sequence>
            <copy>...</copy>
            ... ...
            <copy>...</copy>
            <reply ... />
         </sequence>
    </onMessage>
    <target linkName="start-to-1" />
  </pick>
  <pick>
    <onMessage="I">
         <sequence>
            <copy>...</copy>
            ... ...
            <copy>...</copy>
            <reply ... />
         </sequence>
    </onMessage>
    <target linkName="start-to-2" />
  </pick>
</process>
```

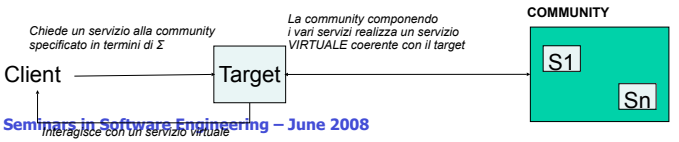
WSCE
A WEB SERVICE (AUTOMATIC) COMPOSITION TOOL

Composizione Automatica (1)

Comunità di servizi:
1. Σ insieme finito di azioni atomiche
2. Insieme di servizi specificato in termini di Σ

Esporta le proprie azioni in termini di Σ

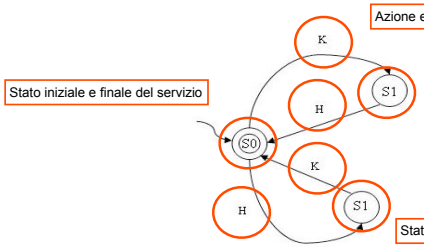
Servizi
- Servizi che delegano proprie azioni ad altri servizi (Servizi Composti)
- Servizi che non delegano proprie azioni ad altri servizi (Servizi Semplici)



Composizione Automatica (2)

Composizione basata su il **BEHAVIOR** di un servizio

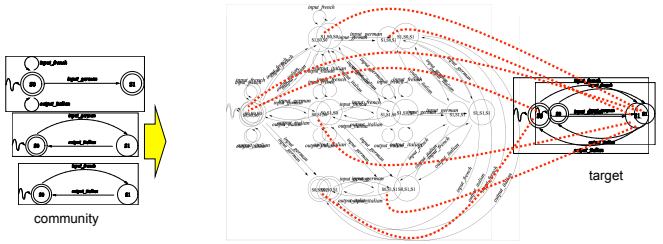
Schema di un servizio:
Modelliamo il servizio con un transition system



Lo schema di un servizio :

- Permette di capire l'ordine con cui le azioni devo essere eseguite;

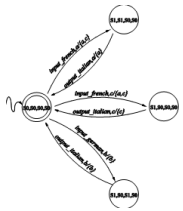
Costruzione del community TS, come **prodotto asincrono** dei TS della community



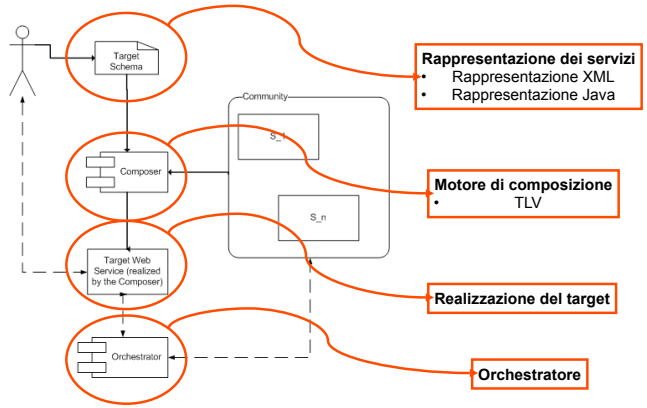
Verifica della relazione di simulazione tra il community TS e il TS del target

TEOREMA
Esiste una relazione di simulazione tra il community TS e il target TS se e solo se esiste una composizione del target rispetto agli available

Estrazione dell'Orchestrator Generator dalla simulazione massima



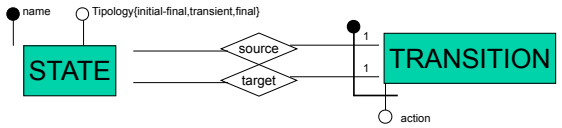
Architettura Astratta del Tool



A TS is composed by: **STATE** and **TRANSITION**

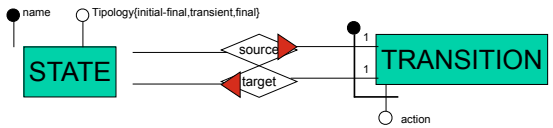
A **STATE** has a *name*(primary key) and a *tipology* (initial-final, transient, final)

A **TRANSITION** has an *action*, and exist iff there are a source and a target state



REQUIREMENTS:

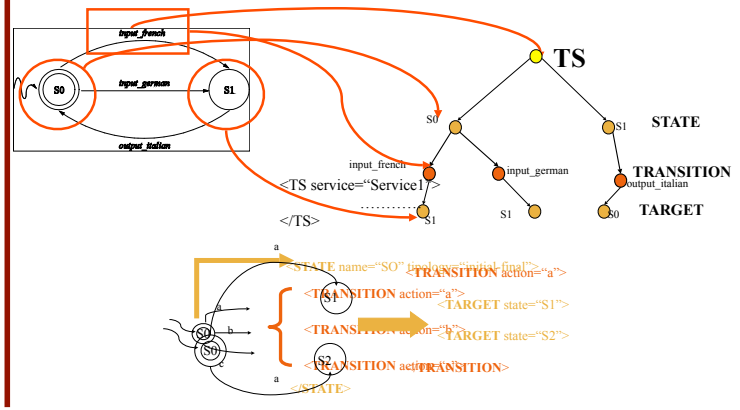
1. easy to write for the user;
2. easiness to check the state of the transition system;
3. easiness to check the transitions;
4. compact: it has to show only the important things.



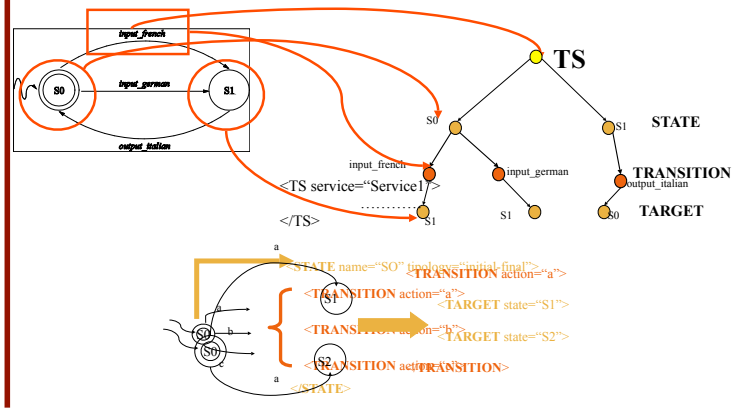
We navigate the schema from a state, through relation source, to its transitions, and from the transitions , through relation target, to the target state reached by such transition.

1. *Natural view of the TS (1,2,3)*
2. *restrained depth of the tree of the document (4)*

Web Service Transition System Language – XML Representation of the Behaviors



Web Service Transition System Language – XML Representation of the Behaviors



WS-TSL XSD

```
<xsd:complexType name="TSType">
   <xsd:sequence>
        <xsd:element name="STATE" type="ns:StateType"
                        minOccurs="1" maxOccurs="unbounded">
        </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="service" type="xsd:string"
        use="required"/>
</xsd:complexType>
```


WS-TSL XSD

```
<xsd:complexType name="StateType">
   <xsd:sequence>
        <xsd:element name="TRANSITION" type="ns:TransitionType"
                maxOccurs="unbounded">
        </xsd:element>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
    <xsd:attribute name="tipology" type="ns:StateTipology" use="required"/
   >
</xsd:complexType>

<xsd:simpleType name="StateTipology">
        <xsd:restriction base="xsd:string">
                <xsd:enumeration value="initial-final"/>
                <xsd:enumeration value="final"/>
                <xsd:enumeration value="transient"/>
        </xsd:restriction>
</xsd:simpleType>
```

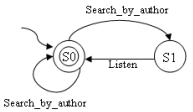

WS-TSL XSD

```
<xsd:complexType name="TransitionType">
        <xsd:sequence>
                <xsd:element name="TARGET" type="ns:Target"
                        minOccurs="1" maxOccurs="unbounded">
                </xsd:element>
        </xsd:sequence>
        <xsd:attribute name="action" type="xsd:string"
                use="required"/>
</xsd:complexType>

<xsd:complexType name="Target">
         <xsd:attribute name="state" type="xsd:string"
                use="required"/>
</xsd:complexType>
```

WS-TSL XSD

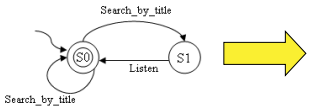

```
<xsd:element name="TS" type="ns:TSType">

        <xsd:unique name="uniqueState">
                <xsd:selector xpath="./STATE" />
                <xsd:field xpath="@name"></xsd:field>
        </xsd:unique>

        <xsd:key name="stateKey">
                <xsd:selector xpath="./STATE" />
                <xsd:field xpath="@name"></xsd:field>
        </xsd:key>

        <xsd:keyref name="transitionTarget" refer="ns:stateKey">
                <xsd:selector xpath="./STATE/TRANSITION/TARGET" />
                <xsd:field xpath="@state"></xsd:field>
        </xsd:keyref>

</xsd:element>
```


Example (1)



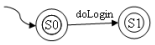

```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="SearchByAuthorND">

    <STATE name="S0" tipology="initial-final">
        <TRANSITION action="SearchByAuthor">
            <TARGET state="S1"></TARGET>
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

    <STATE name="S1" tipology="transient">
        <TRANSITION action="Listen">
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

</TS>
```

Example (2)




```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="SearchByTitleND">

    <STATE name="S0" tipology="initial-final">
        <TRANSITION action="SearchByTitle">
            <TARGET state="S1"></TARGET>
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

    <STATE name="S1" tipology="transient">
        <TRANSITION action="Listen">
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

</TS>
```

Example (3)

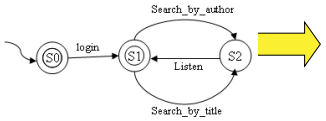



```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="Login">

    <STATE name="S0" tipology="initial-final">
        <TRANSITION action="doLogin">
            <TARGET state="S1"></TARGET>
        </TRANSITION>
    </STATE>

    <STATE name="S1" tipology="transient"/>

</TS>
```

Example (4)

```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="ListenService">

    <STATE name="S0" tipology="initial-final">
        <TRANSITION action="Listen">
            <TARGET state="S0"></TARGET>
        </TRANSITION>
    </STATE>

</TS>
```

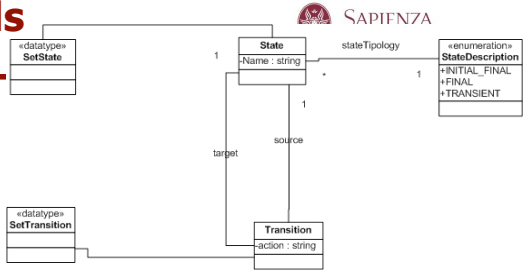
Example (5)

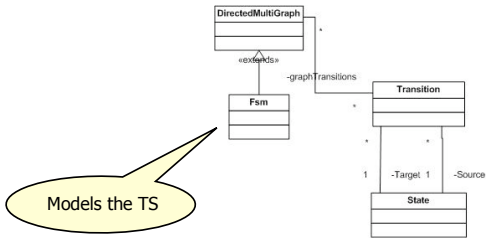


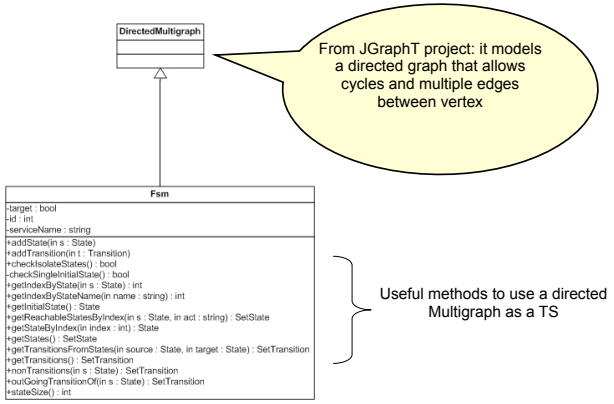

```
<TS
    xmlns='http://www.dis.uniroma1.it/WS-TSL'
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xsi:schemaLocation='http://www.dis.uniroma1.it/WS-TSL

   service="Test10">

 <STATE name="S0" tipology="initial-final">
        <TRANSITION action="DoLogin">
            <TARGET state="S1"></TARGET>
        </TRANSITION>
  </STATE>

  <STATE name="S1" tipology="final">
        <TRANSITION action="SearchByAuthor">
            <TARGET state="S2"></TARGET>
        </TRANSITION>
        <TRANSITION action="SearchByTitle">
            <TARGET state="S2"></TARGET>
        </TRANSITION>
    </STATE>

    <STATE name="S2" tipology="transient">
        <TRANSITION action="Listen">
            <TARGET state="S1"></TARGET>
        </TRANSITION>
    </STATE>

</TS>
```

Java Details

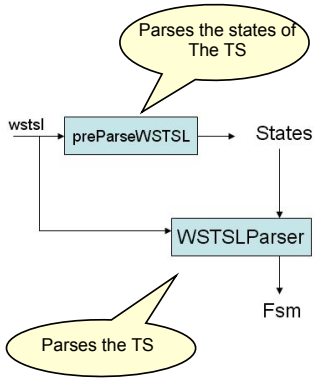
The domain







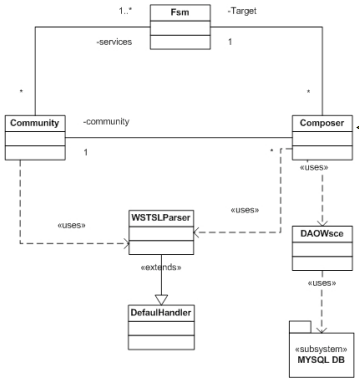
From WSTSL to Java



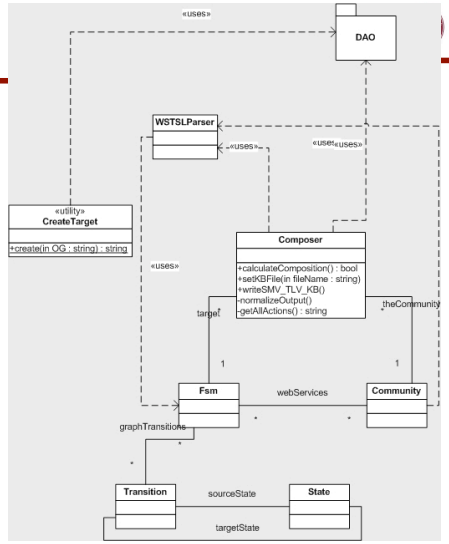
The parsing is performed with SAX

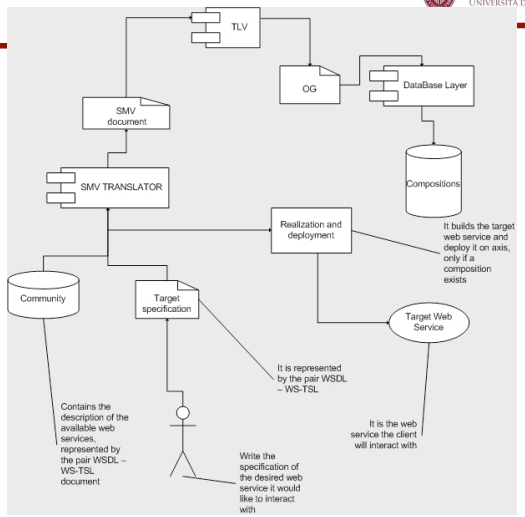
The need of a preparsing is due to the nature of WSTSL documents

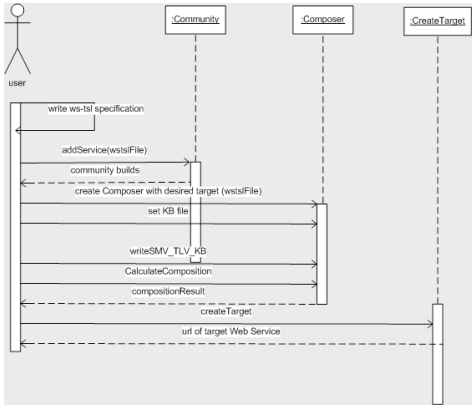
Details of the Engine



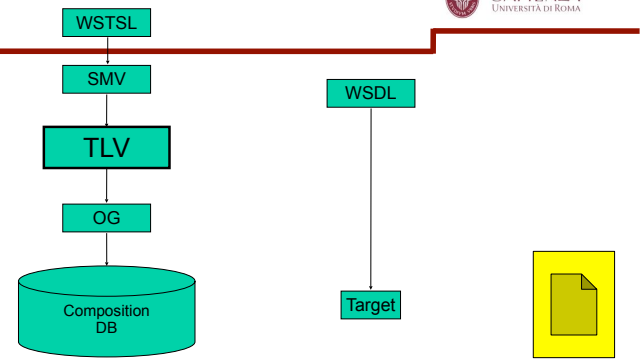
It wraps TLV







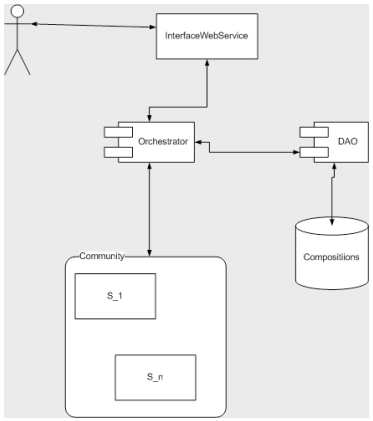
Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma



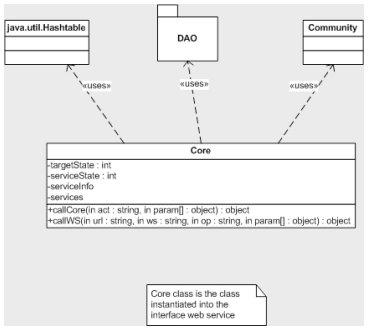
Dall'Orchestrator Generator estraiamo una composizione e la inseriamo in una tabella relazionale per essere acceduta dall'orchestratore

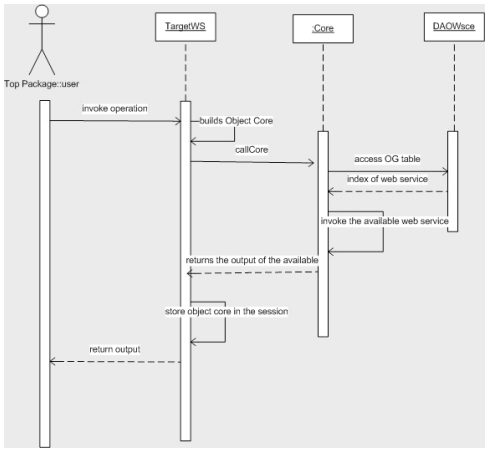
Il target è il Web service d'interfaccia con il client

Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma

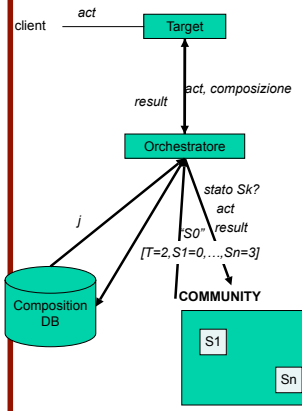


Dipartimento di Informatica e Sistemistica Antonio Ruberti
Sapienza Università di Roma





Orchestratore



1. Il client invoca un'operazione sul target
2. Il target invoca l'operazione sull'orchestratore
3. L'orchestratore chiede lo stato agli available
4. Costruisce la query per sapere chi deve effettuare l'operazione
5. Invoca l'available
6. Restituisce l'output al target e da questi al client

Il passo **3**:
- Mi permette di interagire con servizi parzialmente controllabili
- Necessità di aggiungere un informazione semantica ai servizi

Ogni servizio ha un'operazione aggiuntiva, getStatus, con la quale è possibile sapere lo stato del servizio in un certo istante t

Wrap-up

- Basic SOA needs to be modified for using WSCE
  1. Services should expose their behaviors
     - WS-TLS
  2. Services should offer new operations in their interface for being observed
     - `getStatus()`