An Introduction to Service Oriented Architecture and Web Services

Distributed Systems L-A

Sistemi Distribuiti L-A

Andrea Santi

a.santi@unibo.it

Ingegneria Due

Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2009/2010

Outline

1 Disclaimer

2 Service Oriented Architecture

3 Web Services Framework

4 State of art

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

Disclaimer

- This presentation is organized on the base of the first six chapters of [Erl, 2005]
- As well as the other slides used in this course, most of the content of those slide has been readapted from a reference book [Erl, 2005] and integrated with new material according to the personal view of speaker about this topic
- As usual, eventual mistakes/problems are the sole responsible of the lecturer
- For a more comprehensive picture regarding this topic [Erl, 2005] is a must (and recommended) read

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

What is a Service Oriented Architecture?

Confusion related to the term SOA

- SOA is a term causing a lot of confusion in the IT world
- IT professionals, researchers, anyone claim their own interpretation
  - causing a lot of trouble and misunderstanding

A Joke

Two IT professionals were discussing, when one asked the other if his team was building a service-oriented architecture. The individual responded "My architect thinks it's service-oriented, my developers insist it's object-oriented, and my analysts wish it would be more business-oriented. All I can tell you is that it is not what it was before we started building Web services." [Erl, 2005]

The False SOA *vs.* the Real SOA I

The real SOA

- SOA represent a baseline distributed architecture *with no reference to implementation*
- A platform capable of revolutionizing enterprises promoting
    - federation
    - agility
    - cross-platform harmony

The False SOA *vs.* the Real SOA II

The false SOA

- A technical architecture deemed service-oriented is simply one comprised of Web services...
  - ...probably interacting by means of RPC-like invocations
- Common but dangerous assumption that leads to the number one mistake
  - mainstream SOA are attainable solely through a deeper investment in the Web services platform $\rightarrow$ false!!!

The False SOA *vs.* the Real SOA III

The real SOA vision

Service-orientation presents an ideal vision of a world in which resources are cleanly partitioned and consistently represented. When applied to IT architecture, service-orientation establishes a universal model in which automation logic and even business logic conform to this vision. This model applies equally to a task, a solution, an enterprise, a community, and beyond [Erl, 2005].

The False SOA *vs.* the Real SOA IV

Consequence relates to the adoption of SOA

- Past technical and philosophical disparities are blanketed away
  - by new layers of abstraction that introduce a standards for representing logic and information
- Need of new first-class abstractions and design tools
  - Majors vendors are already promoting support for SOA-some solution/platform... Why, then, is the false SOA so common?

False SOA *vs.* Real SOA: summing up

- The rise of the false SOA has distorted the vision of the ideal SOA
    - The false SOA is so divergent from the *true path of service-orientation*, that it reinforces SOA anti-patterns (RPC, strong control-coupling etc).
- Real SOA requires real change, real foresight, and real commitment
    - A real understanding of service-orientation is required
    - How to shape technical architecture into SOA?
    - Concrete step-by-step processes for realizing it
- SOA makes some impositions
    - A change in mindset is required: business logic now needs to be viewed within a service-oriented context
    - Solution now need to be engineered in a service-oriented fashion
    - Is required a technical architecture capable of hosting service-oriented automation logic

GOALS of this short lesson

After this short lesson/lecture a student should

- Understand SOA, service-orientation, Web services...
    - ...and the **differences** between these concepts

- Learn the basics for building SOA with Web services

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

So, what is really a Service Oriented Architecture?

A more *formal* definition

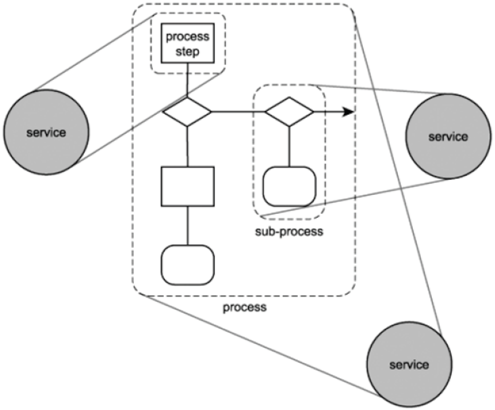
SOA can be defined as an *open, agile, extensible, federated, composable* architecture *comprised of autonomous, QoS-capable, vendor diverse, inter-operable, discoverable*, and *potentially reusable* services [Erl, 2005]

Main features of the service-oriented architectural model

- A service encapsulate a unit of logic within a certain context
- Loose coupling and message-based interactions
- Composability
- Reusability
- Multi-vendor support and interoperability

How service encapsulate logic?

- Services encapsulate logic within a distinct context
- This context can be specific to a business task, a business entity, or some other logical grouping

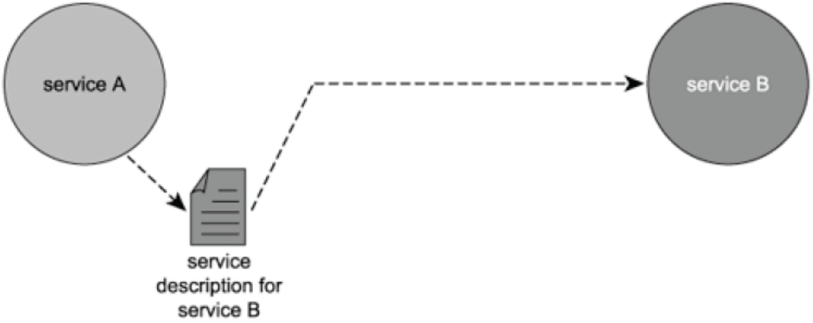


How service relates? 1/2

- Services relationship is based on an understanding that for services to interact, they must be aware of each other
- This awareness is achieved through the use of *service descriptions*
- A service description establishes (at least)
    - The name of the service
    - The data expected and returned by the service
- The manner in which services use service descriptions results in a relationship classified as *loosely coupled*

How service relates? 2/2

- Service A is aware of service B because service A is in possession of service B's service description
- Having access to service B's service description, service A has all of the information it needs to communicate with service B

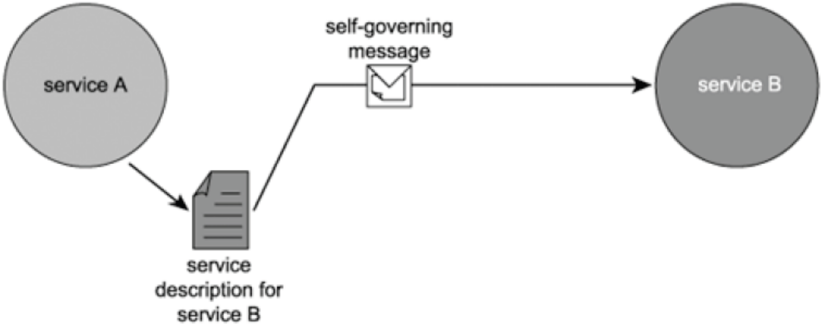


How service communicates? 1/2

- Services communicates by means of proper exchanges of messages
- After a service sends a message on its way, it loses control of what happens to the message thereafter
  - Messages exist as *independent units of communication*
  - Messages, like services, should be *autonomous*
  - Messages must be outfitted with enough intelligence to *self-govern* their parts of the processing logic

How service communicates? 2/2

A simple communication example



How design all those things? 1/2

So, what's so different here?

This architecture appears similar to past distributed architectures that support messaging and a separation of interface from processing logic

The differences reside in..

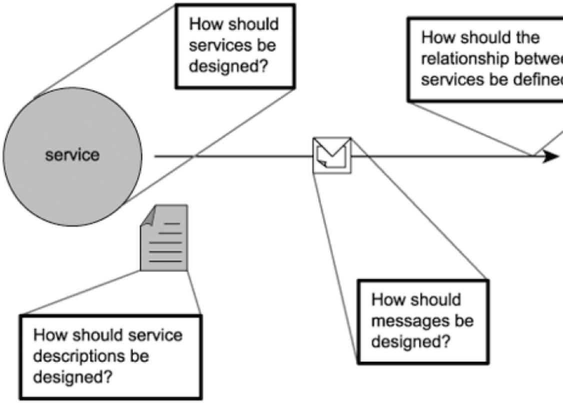
What distinguishes SOA is how its three core components (services, descriptions, and messages) are designed

- This is where service-orientation comes in!

Proper service-orientation principle

Like object-orientation, service-orientation has become a distinct design approach which introduces commonly accepted principles that govern the engineering of an application

How design all those things? 2/2


How should services be designed?
How should the relationship between services be defined?
service
How should service descriptions be designed?
How should messages be designed?

Service orientation principles 1/2

- Services are reusable
  - Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse
- Services share a formal contract
  - For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange
- Services are loosely coupled
  - Services must be designed to interact without the need for tight, cross-service dependencies
- Services abstract underlying logic
  - The only part of a service that is visible is what is exposed via the service contract. Underlying logic is invisible and irrelevant to service

Service orientation principles 2/2

- Services are composable
    - This allows logic to be represented at different levels of granularity and promotes reusability
- Services are autonomous
    - The service has control within its boundary and is not dependent on other services for its governance
- Services are stateless (*more or less*)
    - State information can impede their ability to remain loosely coupled: i.e services should be designed to maximize statelessness
- Services are discoverable (*more or less*)
    - Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

What is the Web Service Framework?

A short premise

SOA is intrinsically reliant on Web services so much so that Web services concepts and technology used to actualize service-orientation have influenced a number of the SOA characteristics identified before [Erl, 2005]

So, what is the Web Service Framework?

- Reference framework that provides a *concrete implementation* of the service-oriented architecture
  - However other implementation are possible
- Must be guaranteed the the right adoption of the principle and the features of SOA

Therefore, SOA and Web Services are not synonyms!!!

- The former it's a *definition* of an architecture (principles, features...)
- The latter is a *concrete implementation* of the service-oriented architectural model

Mapping of SOA concepts into the WS framework

The framework is conceptually in alignment with the main principles of service-orientation

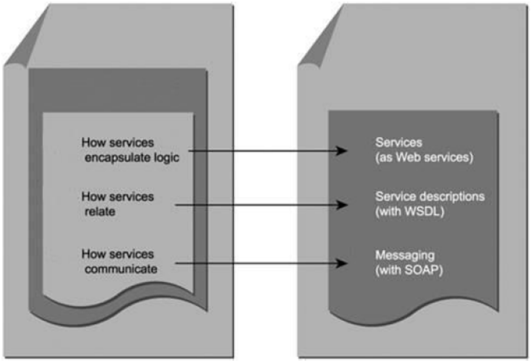

How services encapsulate logic
How services relate
How services communicate
Services (as Web services)
Service descriptions (with WSDL)
Messaging (with SOAP)

Figure: Mapping of SOA concepts into the WS framework [Erl, 2005]

Services (as Web Services)

Web Service (WS): main features

- Technology abstraction used for concretely implement a *service* in a service-oriented fashion
- It can be designed to duplicate the behavior and functionality found in proprietary distributed systems, or it can be designed to be fully SOA-compliant
  - therefore Web services are not necessarily inherently service-oriented

A Web Service can be associated with...

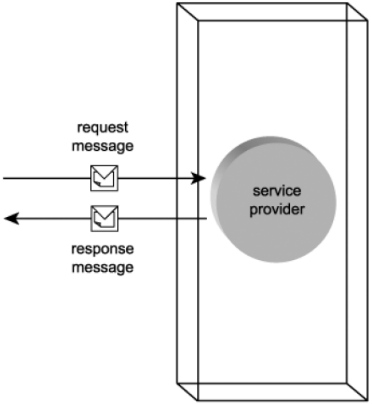
A service role — runtime classification depending on its responsibility in a given scenario (initiator - requestor - intermediary)

A service model — permanent classification depending the role played by the WS into an application (broker - utility service...)

Service provider role

A WS recipient of a request message is classified as a service provider

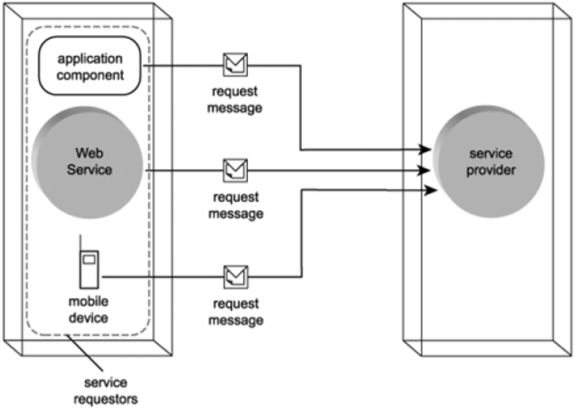
- The WS is invoked by an external source
- The WS provides a published service description (WSDL)



Service requestor role

The sender of a request message is classified as a service requestor

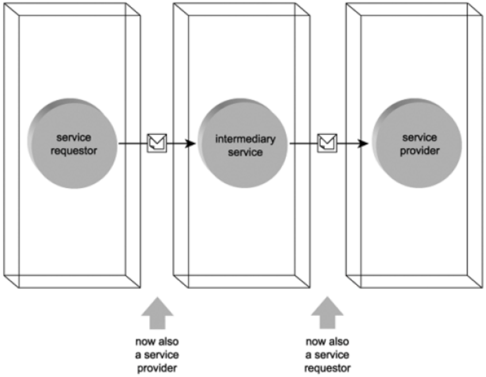
- The WS invokes a service provider by sending it a message
- The WS searches for the most suitable service provider studying available service descriptions



Service intermediary role

A message can be processed by multiple intermediate before its final destination

- Passive intermediaries: simply route messages
- Active intermediaries: route messages to a forwarding destination actively processing/altering the message contents



Simple Object Access Protocol (SOAP) 1/2

Definition

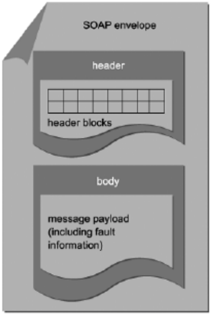
The standard transport protocol for messages processed by Web services

- Originally designed to replace proprietary RPC protocols (i.e. serialization of object)
- Now, despite the name, the purpose is to define a standard message format
  - Important remark: others transport protocols can be used as well
- Extremely flexible and extensible
  - has been revised several times to accommodate more sophisticated features and message structures

Simple Object Access Protocol (SOAP) 2/2

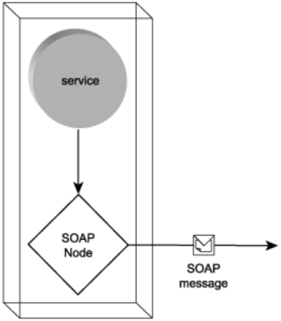
Structure of a SOAP message

envelope	— the message container: house all parts of the message
header	— dedicated to hosting meta-information (used by WS-* specifications, described next)
body	— the message content (i.e. XML-formatted data)



SOAP Nodes

- WS are self-contained units of processing logic, but they are reliant upon a physical communications infrastructure
- Every platform has its own implementation of SOAP communications
- In abstract, the programs that services use to transmit/receive SOAP messages are referred as *SOAP nodes*



Web Service Description Language (WSDL) 1/2

- XML-based language used for defining *service descriptions*
- A WSDL document define
  - The functionalities provided by the service
  - The service behavior

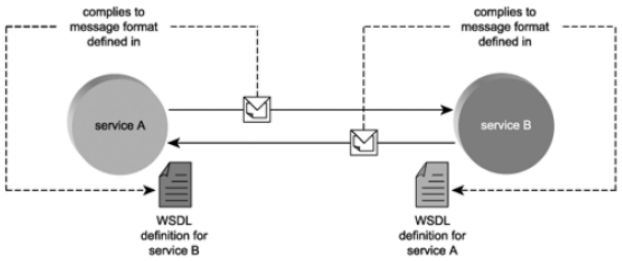

complies to message format defined in
complies to message format defined in
service A
service B
WSDL definition for service B
WSDL definition for service A

Figure: WSDL definitions enable loose coupling between services

Web Service Description Language (WSDL) 2/2

Parts of a WSDL document

A WSDL service description is composed of two parts

- An abstract description
- A concrete description

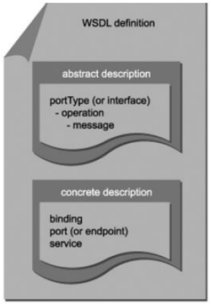


Figure: A WSDL document abstract representation

WSDL abstract description

Abstract description purpose

Establish the interface characteristics of the Web Service without any reference to

- The technology used for realize the Web Service
- The technology used for transmit/receive messages

Abstract description elements

portType	is a high-level view of the service interface by sorting the *messages* a service can process into groups of functions known as *operations*
operation	is a specific action performed by the service
message	is the abstraction used for describe operation's input/output

WSDL concrete description

Concrete description purpose

Establish the physical connection (*binding*) of the WSDL abstract description to a physical transport protocol

Concrete description elements

binding	describes the requirements (i.e. the transport protocol) for establishing a physical connection with the Web Service
port	is the physical address at which a service can be accessed with a specific protocol
service	define the WS name and the set of service *ports* (i.e. all the possible service contact addresses)

An example of WSDL document 1/5

```
<!-- Namespace definition -->
<wsdl:definitions xmlns:wsdl="http://.../wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="..." xmlns:bm="..." xmlns:wsat="..." targetNamespace="...">
<!-- Types definition -->
<wsdl:types>
  <xsd:schema>
    <xsd:complexType name="BookingRequestType">
      <xsd:sequence>
        <xsd:element name="BookingStartDate" type="xsd:date" />
        <xsd:element name="BookingEndDate" type="xsd:date" />
      </xsd:sequence>
    </xsd:complexType>

    <xsd:complexType name="BookingResponseType">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="Available" type="bm:AvailableType" />
          <xsd:element name="NotAvailable" type="bm:NotAvailableType" />
        </xsd:choice>
      </xsd:sequence>
    </xsd:complexType>
    ...
    <!-- XSD element definition based upon the above types definition -->
    <xsd:element name="BookingRequest" type="bm:BookingRequestType" />
    <xsd:element name="BookingResponse" type="bm:BookingResponseType" />
    ...
  </xsd:schema>
</wsdl:types>
```

An example of WSDL document 2/5

```
<!-- Messages definition-->
<wsdl:message name="BookingRequest">
  <wsdl:part name="parameters" element="bm:BookingRequest" />
</wsdl:message>
<wsdl:message name="BookingResponse">
  <wsdl:part name="parameters" element="bm:BookingResponse" />
</wsdl:message>
<wsdl:message name="BookingCancellationRequest">
  <wsdl:part name="parameters" element="bm:BookingCancellationRequest" />
</wsdl:message>
<wsdl:message name="BookingCancellationResponse">
  <wsdl:part name="parameters" element="bm:BookingCancellationResponse" />
</wsdl:message>
<wsdl:message name="SubscribeForDate">
  <wsdl:part name="parameters" element="bm:Subscribe" />
</wsdl:message>
<wsdl:message name="UnscribeForDate">
  <wsdl:part name="parameters" element="bm:Unscribe" />
</wsdl:message>
<wsdl:message name="NotifyDateStatus">
  <wsdl:part name="parameters" element="bm:NotifyDateStatus" />
</wsdl:message>
```

An example of WSDL document 3/5

```
<!-- PortType definition-->
<wsdl:portType name="HotelManagerPortType">

  <wsdl:operation name="BookingOperation">
    <wsdl:input message="bm:BookingRequest" />
    <wsdl:output message="bm:BookingResponse" />
  </wsdl:operation>

  <wsdl:operation name="BookingCancellationOperation">
    <wsdl:input message="bm:BookingCancellationRequest" />
    <wsdl:output message="bm:BookingCancellationResponse" />
  </wsdl:operation>

  <wsdl:operation name="SubscribeForDateOperation">
    <wsdl:input name="SubscribeForDate" message="bm:Subscribe" />
  </wsdl:operation>

  <wsdl:operation name="UnscribeForDateOperation">
    <wsdl:input name="UnscribeForDate" message="bm:Unscribe" />
  </wsdl:operation>

  <wsdl:operation name="NotifyDateStatusOperation">
    <wsdl:output name="NotifyDateStatus" message="bm:NotifyDateStatus" />
  </wsdl:operation>
</wsdl:portType>

<wsdl:portType name="ParticipantPortType">
...
</wsdl:portType>
```

An example of WSDL document 4/5

```
<!-- PorType binding definition -->
<wsdl:binding name="HotelManagerPortTypeBinding" type="bm:HotelManagerPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
  <wsdl:operation name="BookingOperation">
    <soap:operation
      soapAction="http://localhost:8080/wsdl/BookingManager/BookingOperation" />
    <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="BookingCancellationOperation">
    ...
  </wsdl:operation>
  <wsdl:operation name="SubscribeForDateOperation">
    <soap:operation
      soapAction="http://localhost:8080/wsdl/BookingManager/SubscribeForDateOperation" />
    <wsdl:input name="Subscribe">
      <soap:body use="literal" />
    </wsdl:input>
  </wsdl:operation>
  <wsdl:operation name="UnscribeForDateOperation">
    ...
  </wsdl:operation>
  <wsdl:operation name="NotifyDateStatusOperation">
    ...
  </wsdl:operation>
</wsdl:binding>
```

An example of WSDL document 5/5

```
<wsdl:binding name="ParticipantBinding" type="bm:ParticipantPortType">
  <soap:binding transport="http://schemas.xmlsoap.org/soap/http"
    style="document" />
  <wsdl:operation name="PrepareOperation">
    ...
  </wsdl:operation>
  <wsdl:operation name="CommitOperation">
    ...
  </wsdl:operation>
  <wsdl:operation name="RollbackOperation">
    ...
  </wsdl:operation>
</wsdl:binding>

<!-- Service definition -->
<wsdl:service name="HotelManager">
  <wsdl:port binding="bm:ParticipantBinding" name="ParticipantPort">
    <soap:address location="http://localhost:8080/axis2/services/HotelManager" />
  </wsdl:port>
  <wsdl:port binding="bm:HotelManagerPortTypeBinding" name="BookingManagerPort">
    <soap:address location="http://localhost:8080/axis2/services/HotelManager" />
  </wsdl:port>
</wsdl:service>
```

SOAP & WSDL

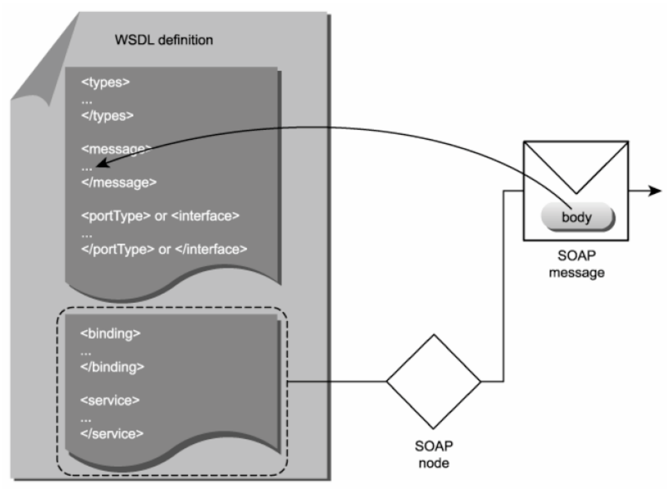


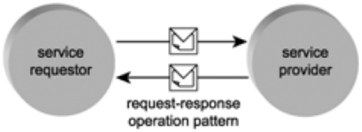
Figure: Relation between a SOAP message and its related WSDL document

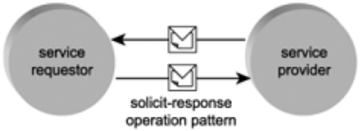
Message Exchange Pattern (MEPs)

- Definition of all the possible communication interaction dynamics between Web Service
- A group of already mapped out sequence for the exchange of messages
- Like design patter in software engineering but oriented to the *message exchange dynamics*

WSDL 1.1 supported MEPs 1/2

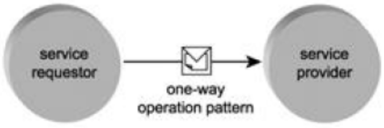
- Request-Response
- Solicit-Response

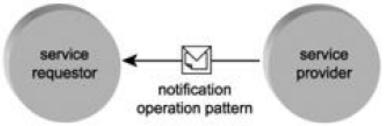




WSDL 1.1 supported MEPs 2/2

- One-way
- Notification





WSDL 2.0 supported MEPs

Old MEPs, but with new names

In-out	equivalent to the Request-Response pattern
Out-in	equivalent to the Solicit-Response pattern
In-only	equivalent to the One-way pattern
Out-only	equivalent to the Notification pattern

New MEPs, introduced by WSDL 2.0

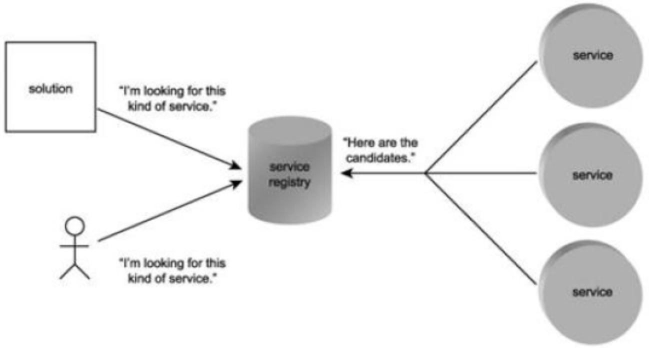
Variations of the basic four MEPs, in addition provides optional in/out message or fault response message

- Robust in-only
- Robust out-only
- In-optional-out
- Out-optional-in

Universal Description Discovery and Integration (UDDI)

OASIS standard that *try* to address the issues related to service discovery and composition

- Functionalities advertising by registering the WS's WSDL into the UDDI registry
- Service requestors search functionalities offered by Web Services simply querying the registry


solution
"I'm looking for this kind of service."
service registry
"Here are the candidates."
service
service
service
"I'm looking for this kind of service."

UDDI problems and limitations

Main problems: no semantics issues are considered!

- Without addressing semantic issues Web Service discovery and composition can not be successfully handled
- UDDI service advertising/discovery only rely upon syntactic aspects
  - Full signature-match for an operation is required
  - Otherwise how infer that a functionality (i.e. a WS operation) such as *rent a vehicle* is the same of *rent a car*?

UDDI isn't so much widespread yet

For taking advance of WS discovery and composition by means of UDDI are required

- A widespread diffusion of the public UDDI registries
- The registration of a high number of WSs

Semantic Web in a nutshell

Tim Berners-Lee vision

I have a dream for the Web [in which computers] become capable of analyzing all the data on the Web - the content, links, and transactions between people and computers. A *Semantic Web*, which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The *intelligent agents* people have touted for ages will finally materialize [Berners-Lee and Fischetti, 1999]

- Goal:*use* and *reason upon* all the available data on the internet automatically
- By extending the current web with knowledge - semantic information - about the content (i.e. data about the data, meta-data)

Semantic Web Service

Introduction

- Researches area, in the ambit of the Semantic Web, that aims to introduce semantics issue into the world of Web Service
- Objective: enable the WS to communicate via *machine-readable* data
- Match regarding *concepts*, *not simply the signature*
    - WS composition/discovery driven by the meaning of the required data/functionalities

Foundations

- Ontologies: rigorous and formal description of a domain (OWL)
- Definition of the WS behavior (OWL-S, WSMO)
    - by means of IOPE (Input, Output, Preconditions, Effects)
- *Software agents* able to find/compose the most suitable WSs w.r.t the user goal

My personal opinion

A key topic but with the current research efforts is only possible grasp the surface of the problem (the same for all the Semantic Web stuff...)

Outline

1. Disclaimer

2. Service Oriented Architecture
   - Introduction
   - SOA fundamentals

3. Web Services Framework
   - Basic concepts
   - Service-orientation and the enterprise
   - Advanced concepts
   - Web Service Tools

4. State of art

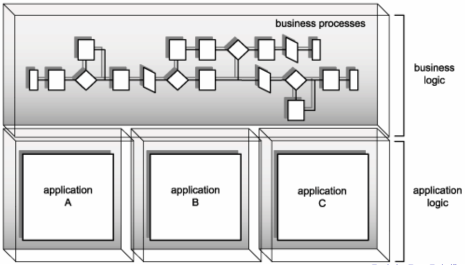
A classical enterprise scenario

Enterprise logic

The logic that defines and drives an enterprise can be divided into

Business logic — implementation of the business requirements that originate from an enterprise's business areas

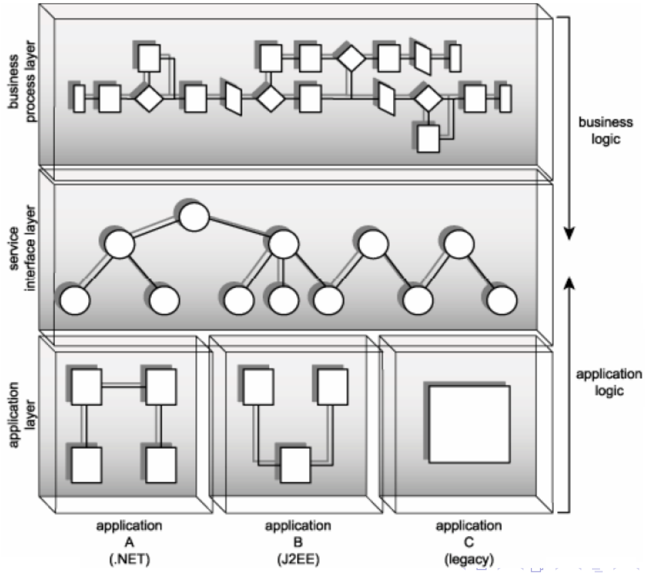
Application logic — structured processes that express business requirements, with constraints, dependencies, and outside influences



SOA impact into an enterprise 1/3

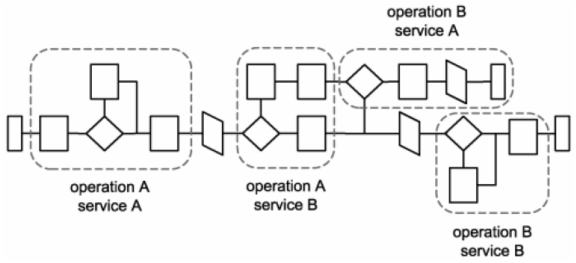
- Service-orientation applies to enterprise logic
  - Establishing an abstraction wedged between business and application layers
- Introduces new concepts that augment the manner in which this logic is represented, viewed, modeled, and shared
- The principles behind service-orientation need a counterpart into the real world environment
  - As new Web Services
  - Embedding legacy code (appropriately refactored)
- Services can be layered
  - Parent services can encapsulate child services
  - This allows the creation of multiple layers of abstraction

SOA impact into a enterprise 2/3



SOA impact into an enterprise 3/3

- Services modularize the enterprise, forming standalone units of logic that exist within a common connectivity layer
- The developer have to find the best modularization/organization solution w.r.t. a certain *enterprise context*
  - Service can cover different units of the business process
  - Complex business task can required the composition/interaction of multiple services



Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

The SOA/WS evolution

The first WS generation introduced the framework building blocks and the basic specifications: WSDL, SOAP, UDDI...

The second generation of Web Service

With the second generation of WS has been introduced a set of specification (WS-*) for the managing of advanced functionalities:

WS-Coordination	provides the rules for coordinating complex activities (AtomicTransactions , BusinessActivities) between WSs
WS-Security framework	is a set of security specifications that provides authentication, authorization, data integrity and so on...
WS-BPEL	define a language for specifying business process behavior based on Web Services
Many others	WS-MetadataExchange, WS-Choreography, WS-Federation..

WS-Coordination

Main features

- Define a general-purpose framework for managing complex activities
- Rooted on a general model for coordinating the common part of different complex activities
  - i.e different coordination activities can be coordinated using the same coordination model
- Aspects related to a particular coordination type are defined into a separated specification

Supported coordination types

Currently only two coordination types are supported

- WS-AtomicTransaction
- WS-BusinessActivities

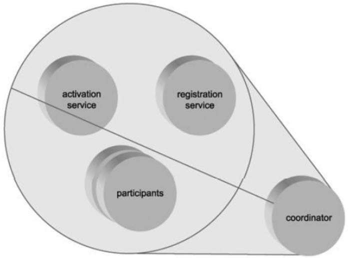
WS-Coordination general model

Service involved

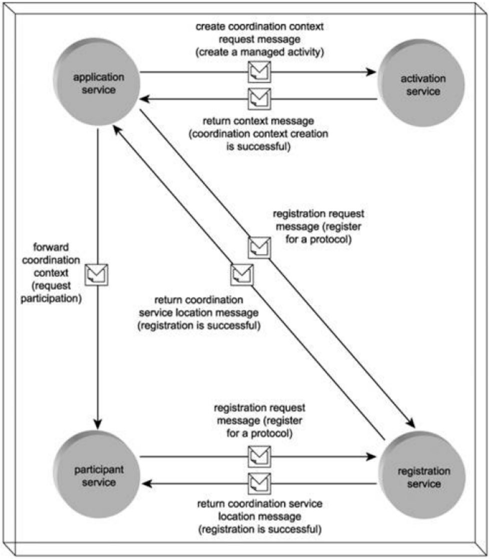
Activation Service responsible of the coordination-context's creation (i.e. the identifier of the coordination activity)

Registration Service register and keep track of the participants of a complex activity

Coordinator Service manages the coordination of an activity w.r.t. a particular coordination type



WS-Coordination dynamics example

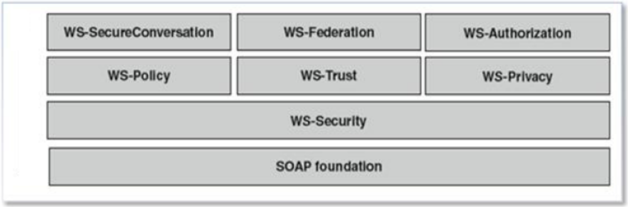


WS-Security framework

A set of WS specifications that address almost all the issues related to Web Service security

Specifications belonging to the security framework

- WS-Security
- WS-Policy
- WS-Trust
- WS-SecureConversation
- Others...



WS-Security and WS-Policy

WS-Security

Enable applications to conduce secure SOAP message exchange ensuring

- Message integrity
- Message confidentiality
- Message authenticity

Rely upon a set of existing specification:XML-Encription, XML-Signature..

WS-Policy

- Define a general purpose model and corresponding syntax to describe the policies of a Web Service...
    - ...also security policies can be defined
- A policy can describe service requirements, capabilities..

WS-Trust and WS-SecureConversation

WS-Trust

Enable applications to construct trusted SOAP message exchanges

- Trust represented through the exchange and brokering of security tokens
- The specification provides a protocol by which: issue, renew and validate security tokens

WS-SecureConversation

Enable secure message exchanges between two or more Web Services

- Built on top of WS-Security and WS-Trust
- Use of security contexts, and derived keys from them, to enable a secure conversation

Web Service Business Process Language (WS-BPEL)

- Orchestration language that provides a means to formally specify business processes and interaction protocols
  - Extends the Web Services interaction model
  - Composition is based on pre-modelled workflow
- Basic activities
  - Invoke, receive, assign
- Structured activities
  - Sequence, flow, foreach

A BPEL business process example

```
<process name="MergedProductionWorkflow">
   <import />
   <partnerLinks />
   <variables />
   <sequence>
      <!-- Step 1: The process-execution layer receives a production order -->
      <receive name="receiveProductionOrder"
         partnerLink="productionManagementPartnerLink"
         portType="prod:ProductionManagementPortType"
         operation="receiveProductionOrder"
         createInstance="yes"
         variable="productionOrder"/>
      <invoke name="getListOfProductionItems"
         partnerLink="productionItemsListPartnerLink"
         portType="list:ProductionItemsListPortType"
         operation="getListOfProductionItems"
         inputVariable="productionOrder"
         outputVariable="productionItems">
      </invoke>
      ...
      <!-- Response back to requesting client application-->
      <reply name="replyPurchaseOrder"
         partnerLink="productionManagementPartnerLink"
         portType="prod:ProductionManagementPortType"
         operation="receiveProductionOrder"
         variable="responseMessage" />
      </sequence>
</process>
```

WS-BPEL: some consideration

- Web Service composition and orchestration realized by means of an offline business plan
- Does this behavior respect the service-orientation principles?
  - Not completely...
- Such an approach (*implicitly*) promotes control coupling between services
  - i.e. Web Service developed already w.r.t a particular role into a orchestration scenario: no reuse

My opinion

Offline and prebuilt business process specifications cannot be the answer

- for building open,scalable and flexible distributed system

Composition and orchestration must be performed at run-time

- still open issues in reaserch

Outline

1. Disclaimer
2. Service Oriented Architecture
   - Introduction
   - SOA fundamentals
3. Web Services Framework
   - Basic concepts
   - Service-orientation and the enterprise
   - Advanced concepts
   - Web Service Tools
4. State of art

Web Service Tools overview

- Brief overview of the architecture of two main Web Service stack implementations
  - Java Metro
  - Apache Axis2 [The Apache Software Foundation, 2004]
- JAX-WS specification [Sun Microsystems, 2004]
  - standard Java-based programming model supported by both

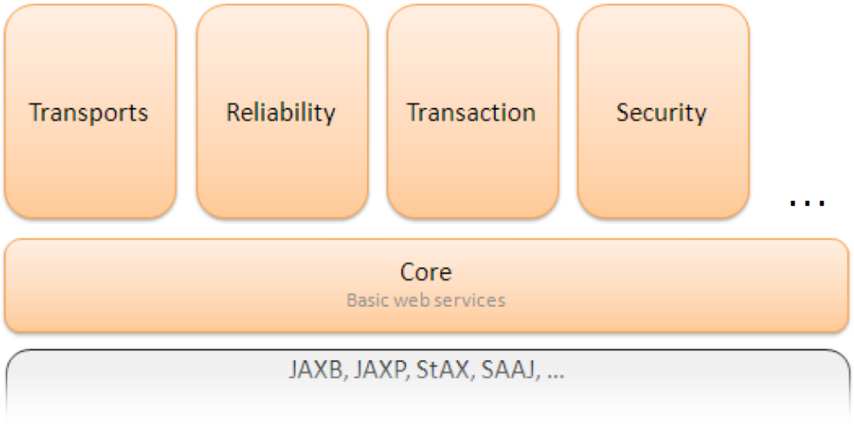
Java Metro

- High-performance, extensible, easy-to-use web service stack.
- Proposed as a *one-stop shop for all your web service needs*
    - from the simplest hello world web service...
    - ... to reliable, secured, and transacted web service that involves .NET services
- Part of the GlassFish community
    - but it can be also used outside GlassFish.

Java Metro and WSIT

- Metro includes WSIT (Web Services Interoperability Technologies)
  - previously known as Project Tango
- It includes implementations of:
  - WS-Trust
  - WS-SecureConversation
  - WS-SecurityPolicy
  - WS-ReliableMessaging
  - WS-AtomicTransactions/Coordination
  - WS-MetadataExchange
  - SOAP over TCP
- Interoperability between the Java platform and Windows Communication Foundation (WCF) (aka Indigo) in .NET 3.0 and .NET 3.5

Metro functionalities



Axis 2

- Java platform for creating and deploying web services applications
  - Born from the Apache implementation of the SOAP specification
- First version: Axis
  - RPC-perspective on Web Services
- New version: Axis 2
  - Web Services in the SOA perspective

Axis 2 features

- Flexible, efficient and configurable architecture
  - supporting SOAP 1.1, SOAP 1.2, REST style of Web services .
  - the same business logic implementation can offer both a WS-* style interface as well as a REST/POX style interface simultaneously
- Modular and XML-oriented
  - it is carefully designed to support the easy addition of plug-in *modules* that extend their functionality for features such as security and reliability
- Modules currently available:
  - WS-ReliableMessaging (Apache Sandesha2)
  - WS-Coordination and WS-AtomicTransaction (Apache Kandula2)
  - WS-Security (Apache Rampart)
  - WS-Addressing (part of Axis2 core)

API Standardizations: JAX-WS 2.0

- It is a specification...
  - so different implementations (e.g. Axis2, Java Metro,..)
- ... of a programming model (= set of API)
  - Java-based
- ...that aims at simplifying application development through support of a standard, annotation-based model to develop Web Service applications and clients in Java
- Document-centric messaging model, replacing the remote procedure call programming model as defined by JAX-RPC
  - SOA perspective

Quick Overview of JAX-WS 2.0

- Simpler way to develop/deploy Web services (w.r.t. JAX-RPC)
  - Plain Old Java Object (POJO) can be easily exposed as a Web service
  - No deployment descriptor is needed - use Annotation instead
  - Layered programming model
- Part of Java SE 6 and Java EE 5 platforms
- Integrated data binding via JAXB 2.0
- Protocol and transport independence

Server-side: two basic ways for building Web Services

- Starting from a WSDL file (top-down approach)
  - Generate classes using ws import
    - WS interface
    - WS implementation skeleton class
  - Add business logic to the WS implementation class
  - Build, deploy, and test
- Starting from a POJO (bottom-up approach)
  - Annotate POJO
  - Build and deploy
  - WSDL file generated automatically

Server-side: an example starting from a POJO

```
import javax.jws.WebService;

@WebService
public class Calculator {
  public int add(int a, int b) {
    return a+b;
  }
}
```

- **@WebService** annotation
  - all public methods become web service operations
- WSDL/Schema generated automatically

Client-side programming 1/2

- The process for creating a web service client application will always start with an existing WSDL document
- Point a tool (e.g. wsimport) at the WSDL for the service
  - wsimport http://example.org/calculator.wsdl
  - wsimport then generates the corresponding Java source code for the described interface
- Call new on the service class
- Get a proxy using a get*ServiceName*Port method
- Invoke any remote operations

Client-side programming 2/2

```
CalculatorService svc = new CalculatorService();
Calculator proxy = svc.getCalculatorPort();
int answer = proxy.add(35, 7);
```

- No need to use factories
- The code is fully portable
- XML is completely hidden from programmer

Principal annotations

@WebService	Marks a Java class as implementing a Web Service, or a Java interface as defining a Web Service interface
@WebMethod	Customises a method that is exposed as a Web Service operation
@WebParam	Customises the mapping of an individual parameter to a Web Service message part and XML element
@WebResult	Customises the mapping of the return value to a WSDL part and XML element

Sessions and states

- Consider a simple Counter Web Service which tracks the user requests
    - each time client calls getCounter(), the service returns the counter after incrementing by 1
- Wrong
    - the JAX-WS only creates one instance of a service class, and have it serve all incoming requests concurrently

```
@WebService
public class Hello {
    int counter = 0;
    public int getCounter() {
        // incorrect – not unique for each client session.
        return counter++;
    }
}
```

Dealing with session 1/2

A solution exploiting resource injection

```
@WebService
public class Hello {

    @Resource
    private WebServiceContext wsContext;

    public int getCounter(){
        MessageContext mc = wsContext.getMessageContext();
        HttpSession session = ((javax.servlet.http.HttpServletRequest)
                   mc.get(MessageContext.SERVLET_REQUEST)).getSession();
        // Get a session property "counter" from context
        if (session == null)
            throw new WebServiceException("No session in WebServiceContext");
        Integer counter = (Integer)session.getAttribute("counter");
        if (counter == null) {
            counter = new Integer(0);
            System.out.println("Starting the Session");
        }
        counter = new Integer(counter.intValue() + 1);
        session.setAttribute("counter", counter);
        return counter;

    }
}
```

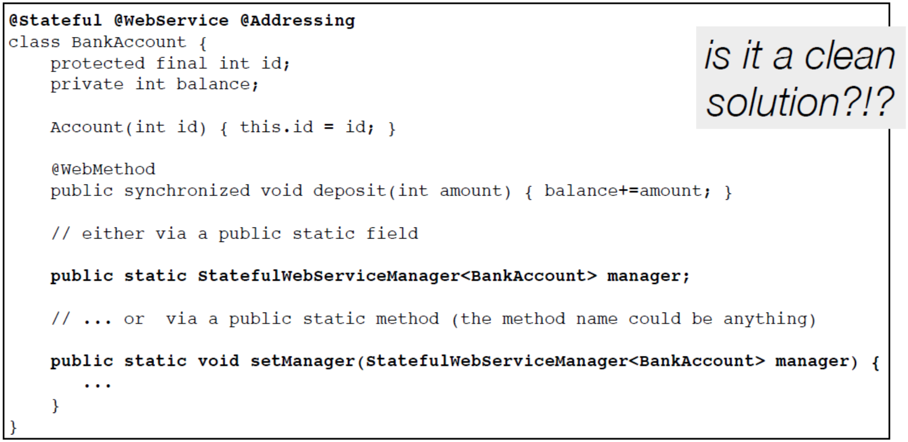
is it a clean solution?!?

Dealing with session 2/2

- JAX-WS uses some annotations to inject the Web Service context and declaring lifecycle methods
  - Web ServiceContext holds the context information pertaining to a requet being served
- Placing **@Resource** annotation on a service endpoint implementation, you can request a resource injection and collect the javax.xml.ws.WebServiceContext interface related to that particular endpoint invocation
- From the WebServiceContext interface, you can collect the MessageContext for the request associated with the particular method call using getMessageContext()

Dealing with state 1/2

Stateful Web Service definition



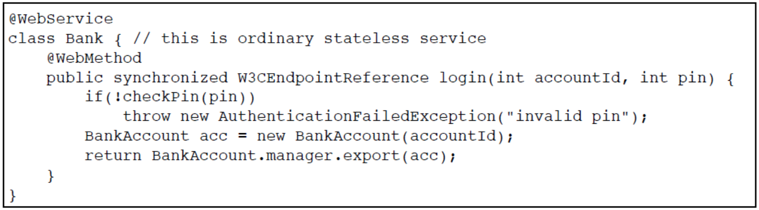
```
@Stateful @WebService @Addressing
class BankAccount {
    protected final int id;
    private int balance;

    Account(int id) { this.id = id; }

    @WebMethod
    public synchronized void deposit(int amount) { balance+=amount; }

    // either via a public static field

    public static StatefulWebServiceManager<BankAccount> manager;

    // ... or  via a public static method (the method name could be anything)

    public static void setManager(StatefulWebServiceManager<BankAccount> manager) {
        ...
    }
}
```

is it a clean solution?!?

Figure: [Pulavarthi, 2006]

Dealing with state 2/2

- The stateful Web Service cannot be directly used
  - Is invoked, using resource injection, by an ordinary stateless Web Service



```
@WebService
class Bank { // this is ordinary stateless service
    @WebMethod
    public synchronized W3CEndpointReference login(int accountId, int pin) {
        if(!checkPin(pin))
            throw new AuthenticationFailedException("invalid pin");
        BankAccount acc = new BankAccount(accountId);
        return BankAccount.manager.export(acc);
    }
}
```

Figure: [Pulavarthi, 2006]

Concluding remarks

- Java Metro & Apache Axis2 technology
    - modular / extensible / customisable architectures
    - supporting the JAX-WS standard programming model
    - not only JAX-WS, but also JAX-WS
- JAX-WS programming model
    - very effective for simple services
    - not so effective for complex services
    - not really *one-stop shop for all your web service needs*

Outline

1 Disclaimer

2 Service Oriented Architecture
- Introduction
- SOA fundamentals

3 Web Services Framework
- Basic concepts
- Service-orientation and the enterprise
- Advanced concepts
- Web Service Tools

4 State of art

The need for a SOA/WS programming model

- SOA is a promising solution for building complex distributed systems, **but does not provide** a unifying programming model to support the core system design and development
- Actually the mainstream proposals are either object-oriented or component-oriented
    - In our opinion this models are deeply inadequate for the design of complex SOA system

Rationale

Agent concepts and technologies could be fruitfully exploited to define a general-purpose programming model to support the design, development, management of complex SOA/WS systems

References to APICe Research Projects about SOA/WS

CArtAgO

- CArtAgO Reference on APICe portal
  - http://alice.unibo.it/xwiki/bin/view/CARTAGO/
- CArtAgO Reference on sourceforge
  - http://sourceforge.net/projects/cartago/

CArtAgO and Friends

- CArtAgO-WS Reference on sourceforge
  - http://sourceforge.net/projects/cartagows/
- CArtAgO bridges references for the integration with different agent platform (such as Jason and so on) can be found on apice portal, too
  - http://alice.unibo.it/xwiki/bin/view/CARTAGO/Integrations

Bibliography I

Berners-Lee, T. and Fischetti, M. (1999).
Weaving the Web: The original design and ultimate destiny of the World Wide Web by its inventor.
Harper San Francisco.

Erl, T. (2005).
Service-Oriented Architecture: Concepts, Technology, and Design.
Prentice Hall PTR, Upper Saddle River, NJ, USA.

Pulavarthi, R. (2006).
Maintaining session with JAX-WS.
`http://ws.apache.org/axis2`.

Sun Microsystems (2004).
JAX-WS reference site.
`https://jax-ws.dev.java.net/`.

Bibliography II

The Apache Software Foundation (2004).
Axis 2 reference site.
http://ws.apache.org/axis2.