SERVICE-orientated COMPONENT-based MODEL

Marco Autili
University of L'Aquila

marco.autili@di.univaq.it
http://www.di.univaq.it/marco.autili

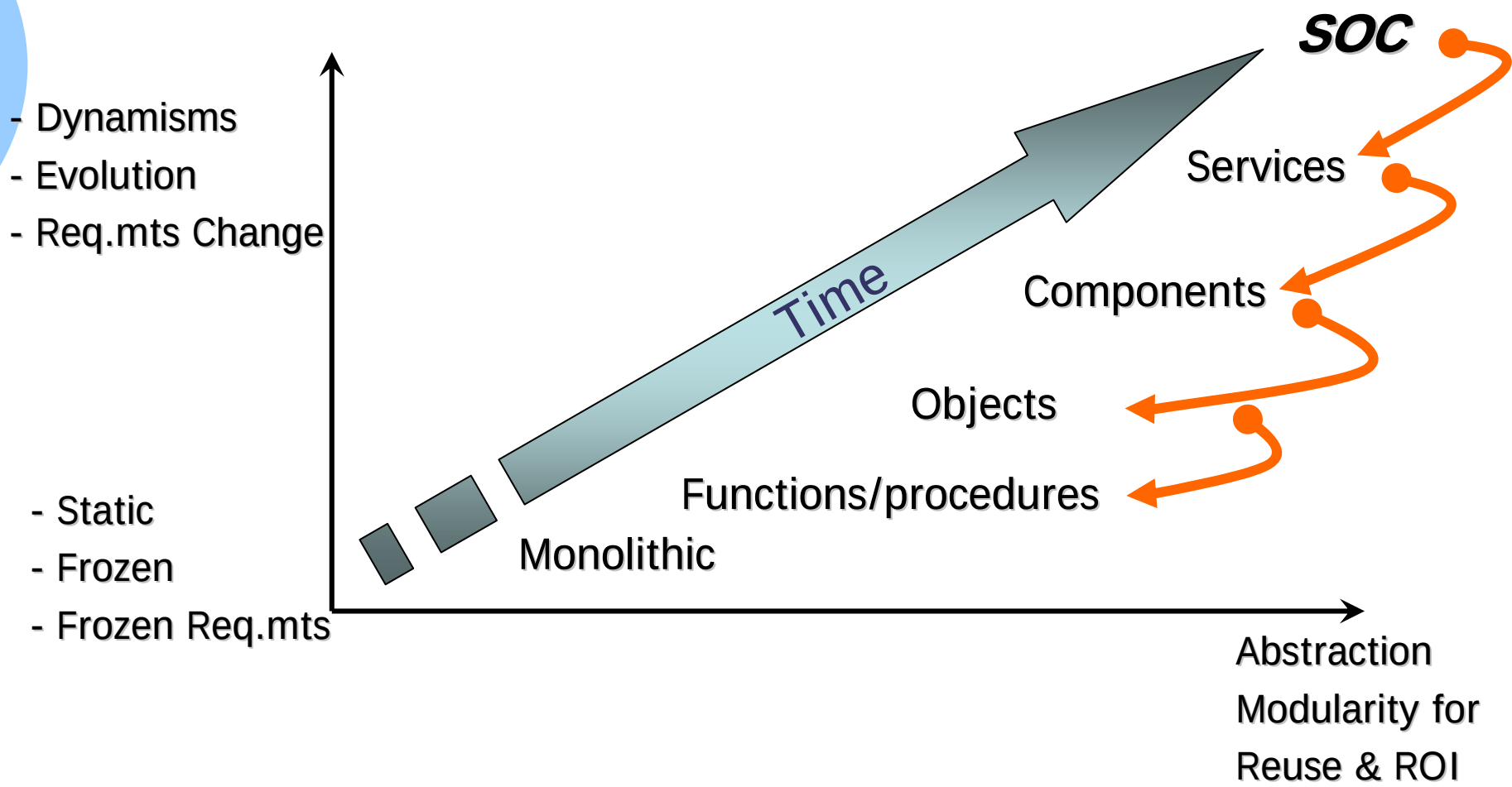
Road Map

- *SOA facts ☺*
- *What are service?*
- *What are software services?*
- *Service Oriented Architecture (SOA)*
- *SOA characteristics*
- *Component orientation VS Service orientation*
- *Web Service in practice*
- *Component orientation + Service orientation*
- *Challenges in creating a service-oriented component model*

The Encyclopedia of SOA FACTS
(http://www.soafacts.com/index.html)

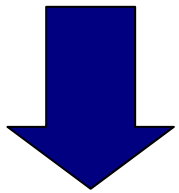
- *SOA is being used in the developing world to solve hunger. Entire populations will be fed on future business value.*
- *SOA can write and compile itself.*
- *SOA is not complex. You are just dumb.*
- *SOA in a Nutshell is 7,351 pages spread over 10 volumes.*
- *One person successfully described SOA completely, and immediately died.*
- *Another person successfully described SOA completely, and was immediately outsourced.*
- *Larry Ellison once died in a terrible accident, but was quickly given SOA. He came back to life, built a multibillion dollar software company, and now flies fighter jets.*
- *SOA is the only thing Chuck Norris can't kill.*

Towards a new computing paradigm



Services

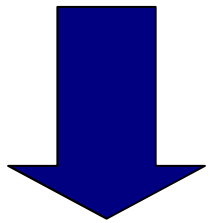
- What are *services* ?
  - One of the abstract and context independent definition is:
    - Services are labors that, if accomplished, produce a non tangible useful benefit (Software Technologies Unit of the European Commission).
- What are *software services* ?
  - Software services are functionalities provided by software applications that supply:
    - computational resources on request;
    - informational resources on request.
  - Functionalities are contractually defined in the *service description* (Software Technologies Unit of the European Commission).



- **S**ervice **O**riented **A**rchitecture (SOA)

SA, CBA and SOA …

- *Software Architecture (SA)*
- *Component Based Architecture (CBA)*
- *Service Oriented Architecture (SOA)*
  - *service designer and developer?*



  - *… a SOA-based system implementation should meet concept and principles of SOA*

The most important aspects of SOA

Service Implementation and Service Provider

Service Contractual Description

HOW

WHAT

Do not worry about HOW ...

... only expect that it will

Why SOA? (1/3)

- SA have attempted to deal with **increasing** levels of software **complexity** through abstraction …
- … "traditional architecture models" seem to be **reaching the limit** of their ability to deal with the "today complexity" of software.

- At the same time, traditional needs of Information Technology (IT) organizations persist:
  - the **need to respond quickly to new requirements** of business;
  - the need to continually **reduce the cost of IT** to the business;
  - the **ability to absorb and integrate** new business partners and new customer sets, etc.
  - the need of a **cost-effective evolution and enhancement** of legacy Enterprise Information System (EIS)

Why SOA? (2/3)

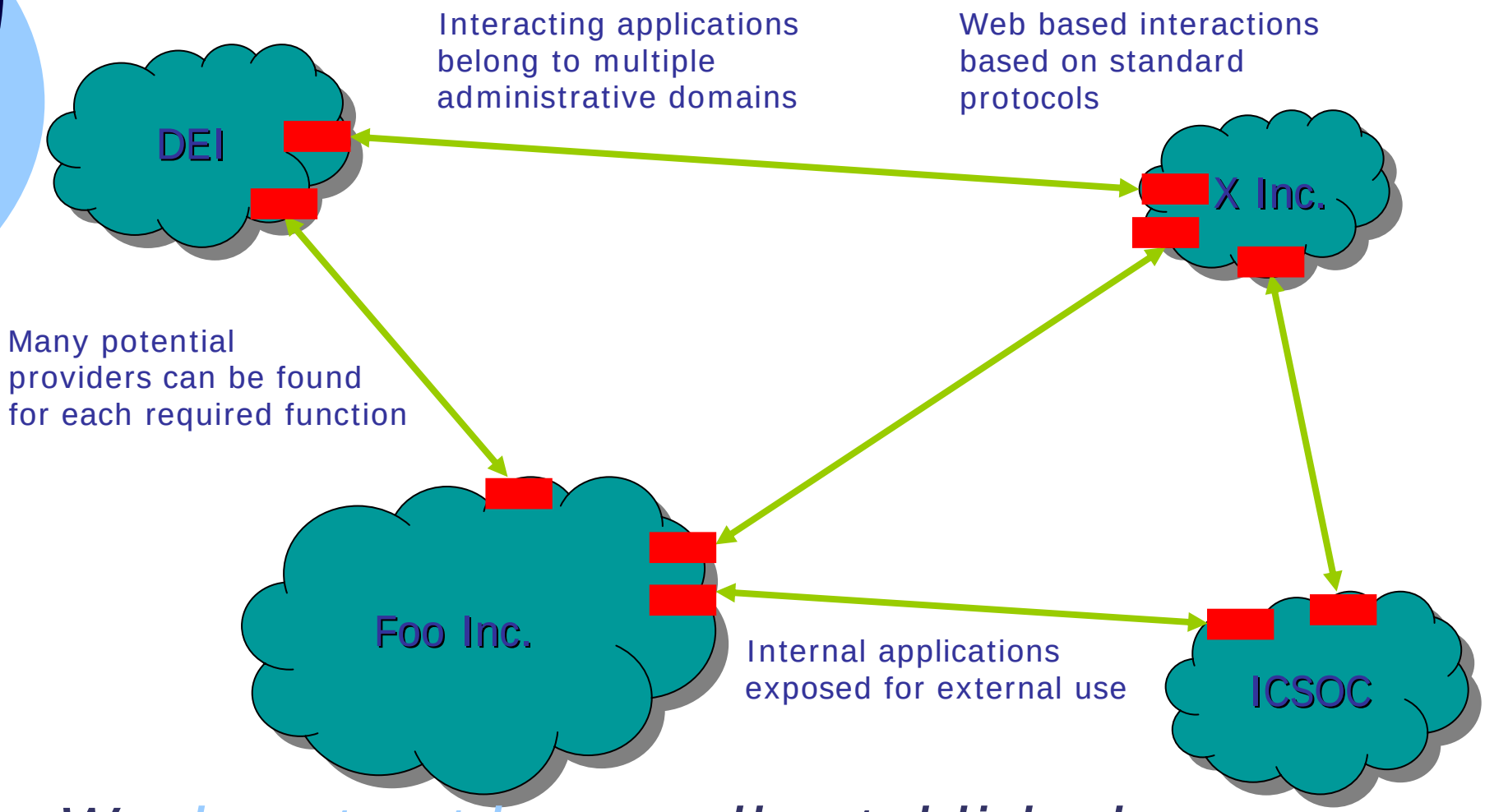
- Corporate management always pushes for:
  - better IT utilization to elaborate and transmit data;
  - greater Return on Investment (ROI):
    - i.e., a financial measure of the relative value of an investment, usually expressed as a percentage, calculated by dividing earnings produced by the amount invested).

- Systems must be developed by taking into account the complexity introduced by heterogeneity, dynamicity, integration, reuse, adaptability, ubiquitous access to the heterogeneous infrastructures:
  - legacy systems must be reused rather than replaced, because with even more constrained budgets, replacement is cost-prohibitive;
  - new business models, so entire IT organizations, applications, and infrastructures must be integrated and absorbed;
  - applications need to be adaptive to the context of use;
  - ...

Why SOA? (3/3)

- We need an architectural framework which allows the **assembly of services** for the **agile** (easy and rapid), and even **dynamic**, **delivery of BUSINESS solutions**.
- Focus on exposing **business functions**, not technology
- SOA is being promoted in the industry as **the next evolutionary step in SA** to help IT organizations meet their evermore complex set of challenges.
- SOA "**could be a 2010 phenomenon**"
  - From: Financial Times, 4 May 2005, Richard Waters: "Plugging together software may soon painless"

(WS-based) Networked organizations



- *We do not yet have well-established software engineering methods* ☹

SOA is not new!

- It is an alternative *loosely-coupled* model to the more traditionally *tightly-coupled* object-/component-oriented models that have emerged in the past decades. In computer science:
  - *Coupling* or dependency is the degree to which each program module relies on each one of the other modules (Wikipedia ???)
  - *Loosely coupled* systems are considered useful when either the source or the destination computer systems are subject to frequent changes (Wikipedia ???)
- The overall design is service-oriented but do not exclude (and that's it) the fact that individual services can themselves be built with object- and/or component-oriented designs.
- The difference lies in the **"interfaces"** themselves. A classic example of a proto-SOA system that has been around for a while is the (CORBA), which defines similar concepts to SOA

SOA is not new!

- … Web Services (as a specific “implementation” of SOA) are different in that it relies on a more recent advance based upon XML.
- By describing interfaces in an XML-based language called WSDL, web services have moved to a more dynamic and flexible interface system than the older IDL found in CORBA.
- Web services aren't the only way to implement SOA.
- CORBA, as just explained earlier is one other way and so are Message-Oriented Middleware (MOM - next slide) systems such as the IBM MQ Series.
- .. to become an architecture model, you need more than just a service description:
  - you need to define how the overall application performs its workflow between services and hence between business activity (orchestration and choreography).

SOA is not new: MOM from Wikipedia … just to know

- **Message-oriented middleware (MOM)** is a client/server infrastructure that increases the interoperability, portability, and flexibility of an application by allowing the application to be distributed over multiple heterogeneous platforms. It reduces the complexity of developing applications that span multiple operating systems and network protocols by insulating the application developer from the details of the various operating system and network interfaces. APIs that extend across diverse platforms and networks are typically provided by the MOM.

- MOM is software that resides in both portions of client/server architecture and typically supports asynchronous calls between the client and server applications. Message queues provide temporary storage when the destination program is busy or not connected. MOM reduces the involvement of application developers with the complexity of the master-slave nature of the client/server mechanism.

- MOM comprises a category of inter-application communication software that generally relies on asynchronous message-passing, as opposed to a request/response metaphor.

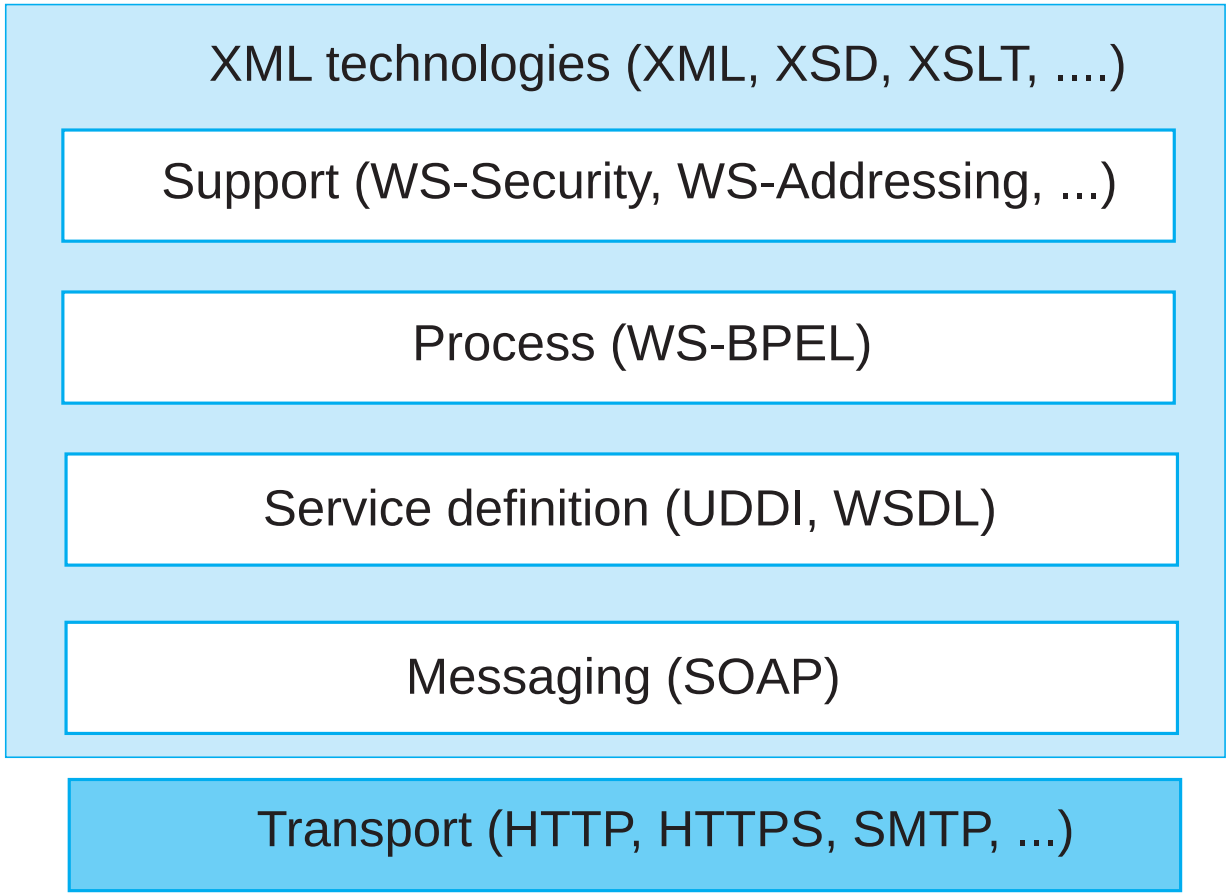
- Most message-oriented middleware depend on a message queue system, but there are some implementations that rely on broadcast or multicast messaging systems.

The success of SOA

- *The success of SOA is being determined by the active standardization process working alongside technical developments to define a unique set of widely accepted standards.*

- *All the major hardware and software companies are committed to these standards (e.g., Web Service XML-based standards).*

Web Service XML-based standards



Guns don't kill people, the SOA WS- stack kills people!* ☺

(http://www.soafacts.com/index.html)

Key standards

- SOAP
  - A message exchange standard that supports service communication
- WSDL (Web Service Definition Language)
  - This standard allows a service interface and its bindings to be defined
- UDDI
  - Defines the components of a service specification that may be used to discover the existence of a service
- WS-BPEL
  - A standard for workflow languages used to define service composition

Component Orientation (1/6)

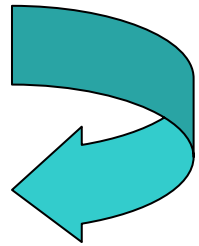
- *CBSE* promotes the *construction* of software applications as *composition* of reusable building *"blocks"* called *components* based on some *specific* standard model.

- The goal of CBSE is the establishment of a common market where to select and buy components.

- *"A software component is a binary unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties" [Szyperski].*

Component Orientation (2/6)

- Component Orientation makes a distinction between *developers* and *assemblers* *(third party)*.



- *Binary Unit* delivering.

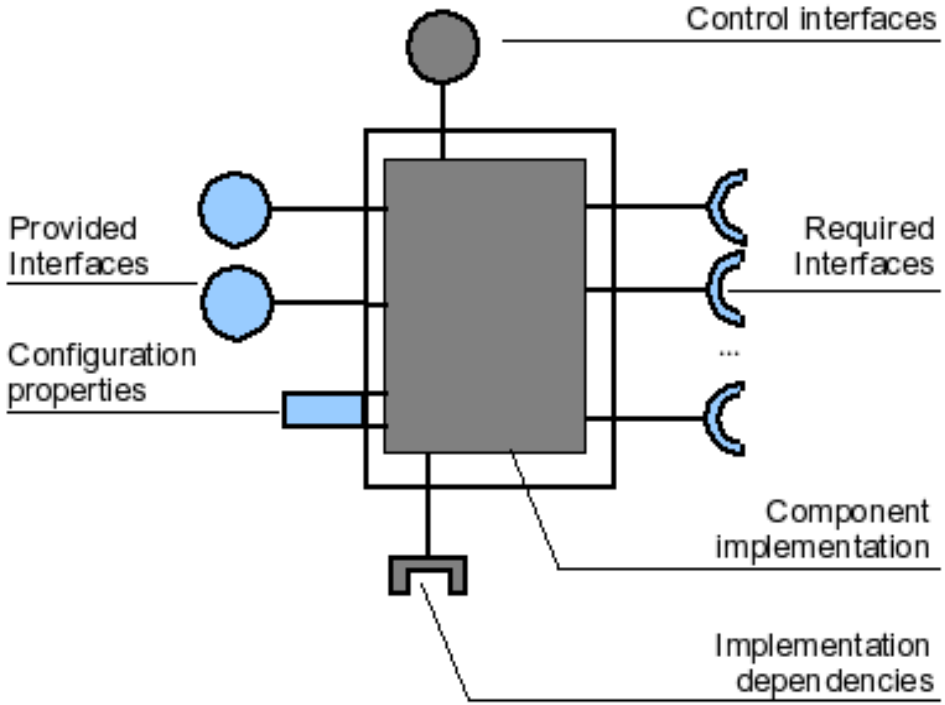
- Application assembly is based on components *physically available* to the assemblers when composing them (due to *explicit (structural) context dependencies*), thus …
  - Integration at *assembling time*:
    - deployment, instantiation and connection.

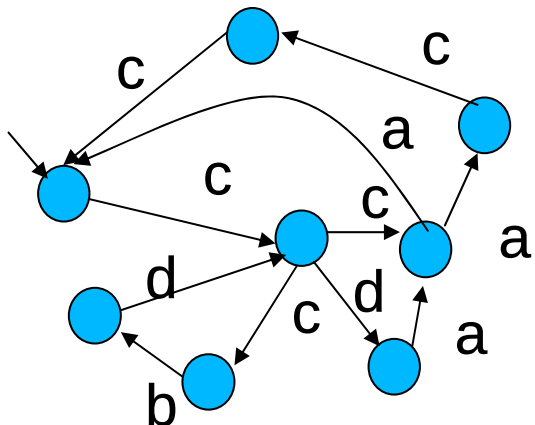
Component Orientation (3/6)

- In the majority industrial component model (COM/DCOM, EJB), component are *similar to classes*.
- Third parties instantiate component and then *connect instances* in some appropriate fashion.
- Component are often *hardly reused* as they are ….
- … *Component Adaptation* promotes the use of adaptors.

Component Orientation (4/6)

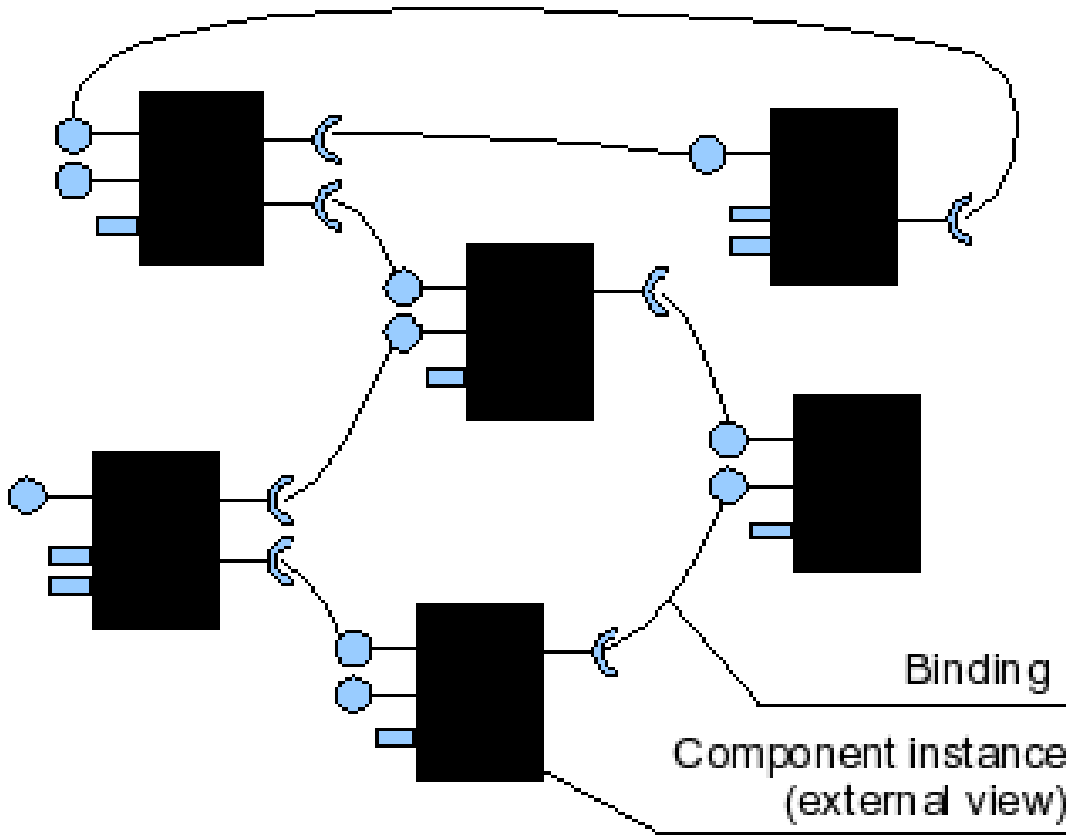
- To support *structural composition*:
  - Component instance external view based on proprietary and/or specific models.
  - Used similarly to the approach of Architecture Description Language (ADL)
  - Behavioral specification can be provided;





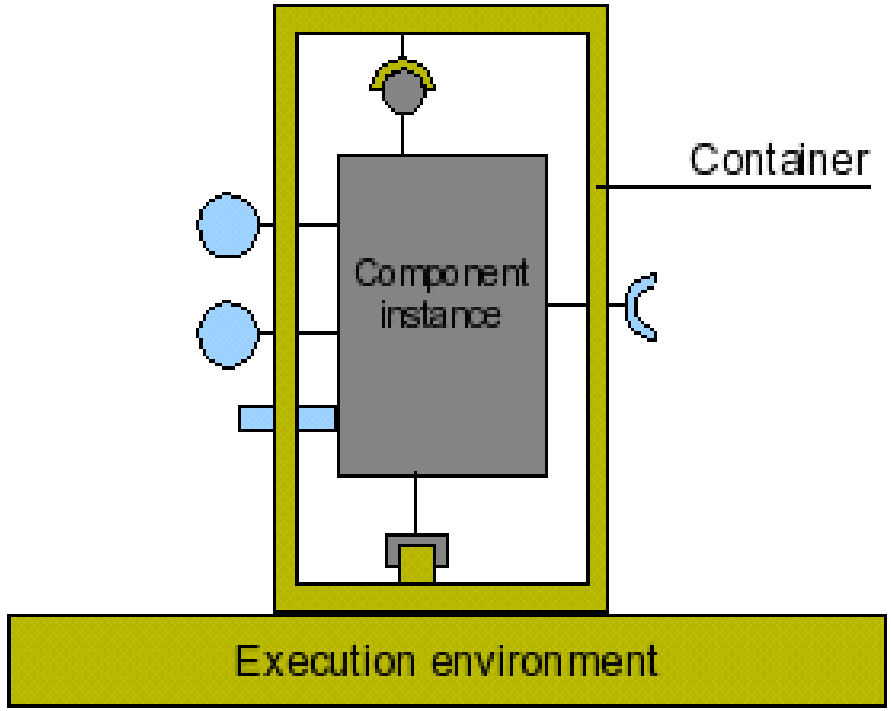
Component Orientation (5/6)

- *Hierarchical composition* is achieved when the *external view* of a component is itself implemented by a composition.



Component Orientation (6/6)

- A component is often delivered and deployed independently as binary code along with its required resources (e.g., images, libraries).
  - *Microsoft Word Processor is such a component*
- Deployment dependencies need to be fulfilled before it can be instantiated.
- Require an execution environment that provides run-time support (e.g., a container)



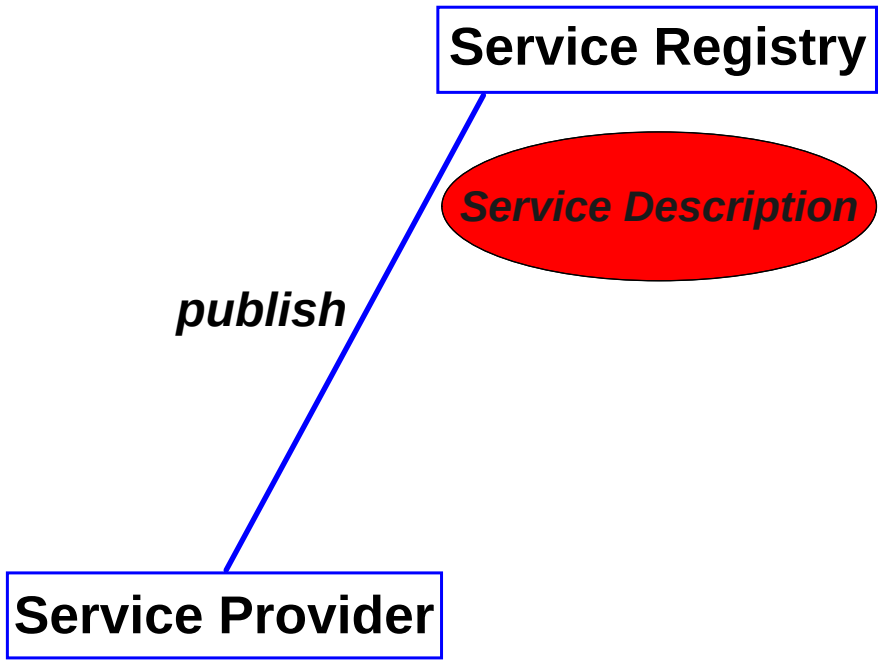
- Run-time support includes life-cycle management and (possibly) handling of extra-functional characteristics (security, persistence, transactions)

Service Orientation (1/5)

- Applications are *assembled* from reusable building *"blocks"*, but in this case the blocks are *services*.
- The goal of "SOA" is the establishment of a common market where to select and buy services.
- Services are *functionalities* defined by contractual descriptions:
  - Services functionality as mix of syntax, semantics and behavior.
- Application assembly is *based only* on *service description.*
- *The most interesting aspects are:*
  - The actual service provider is *located and integrated later* into the application (usually prior or during execution).
  - Thus, the focus is on how the service are described in order to support *dynamic discovery*.

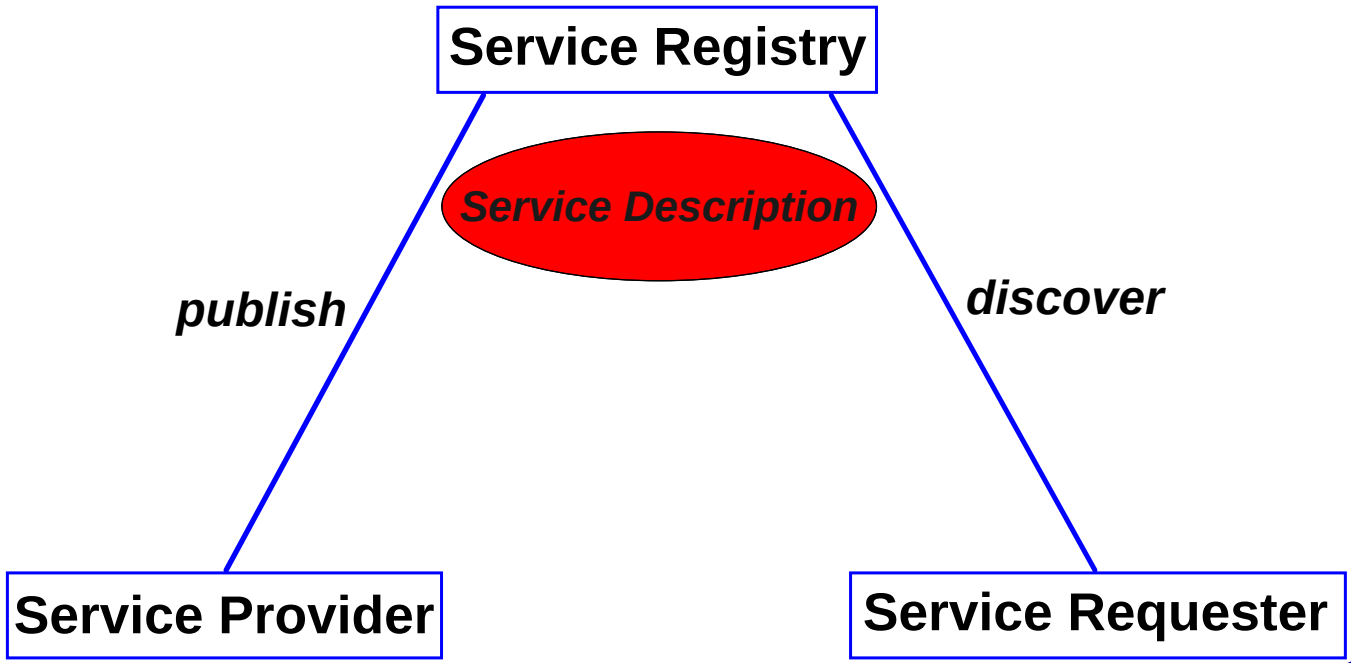
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - Service Providers *publish* their *service description* into a service registry …



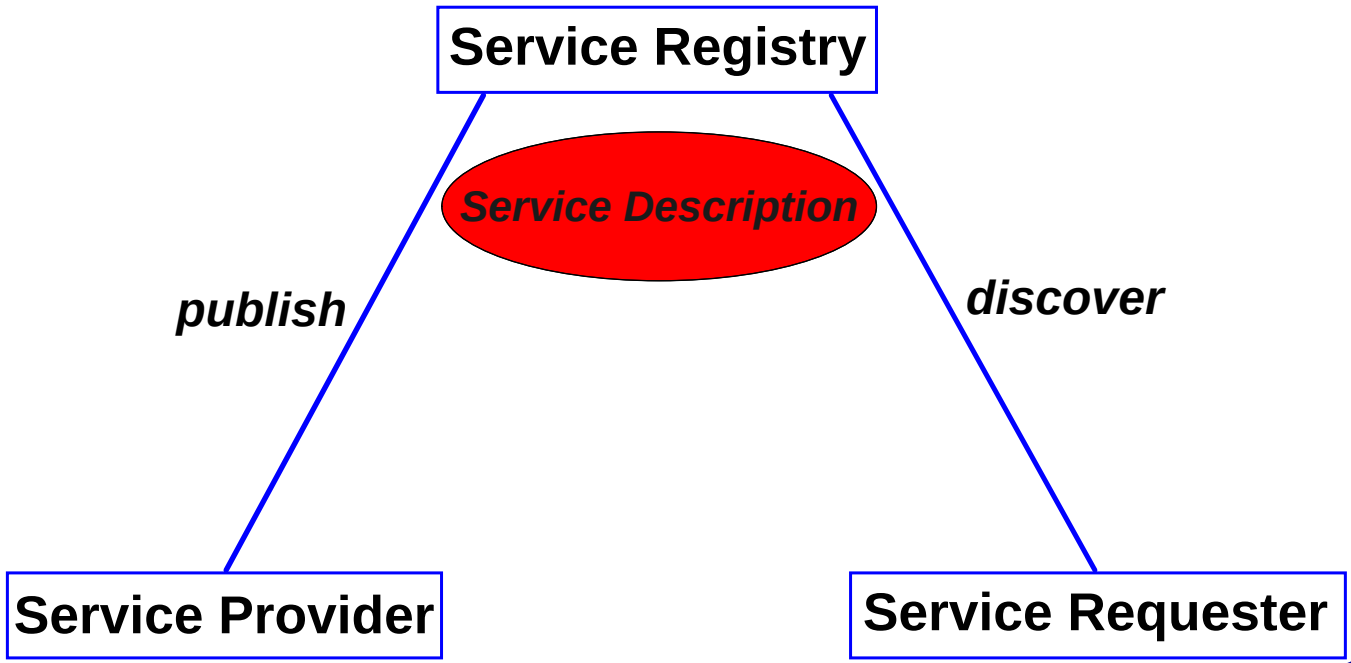
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - Service Requestors interrogate the registry for a particular service description to *discover* service providers ...



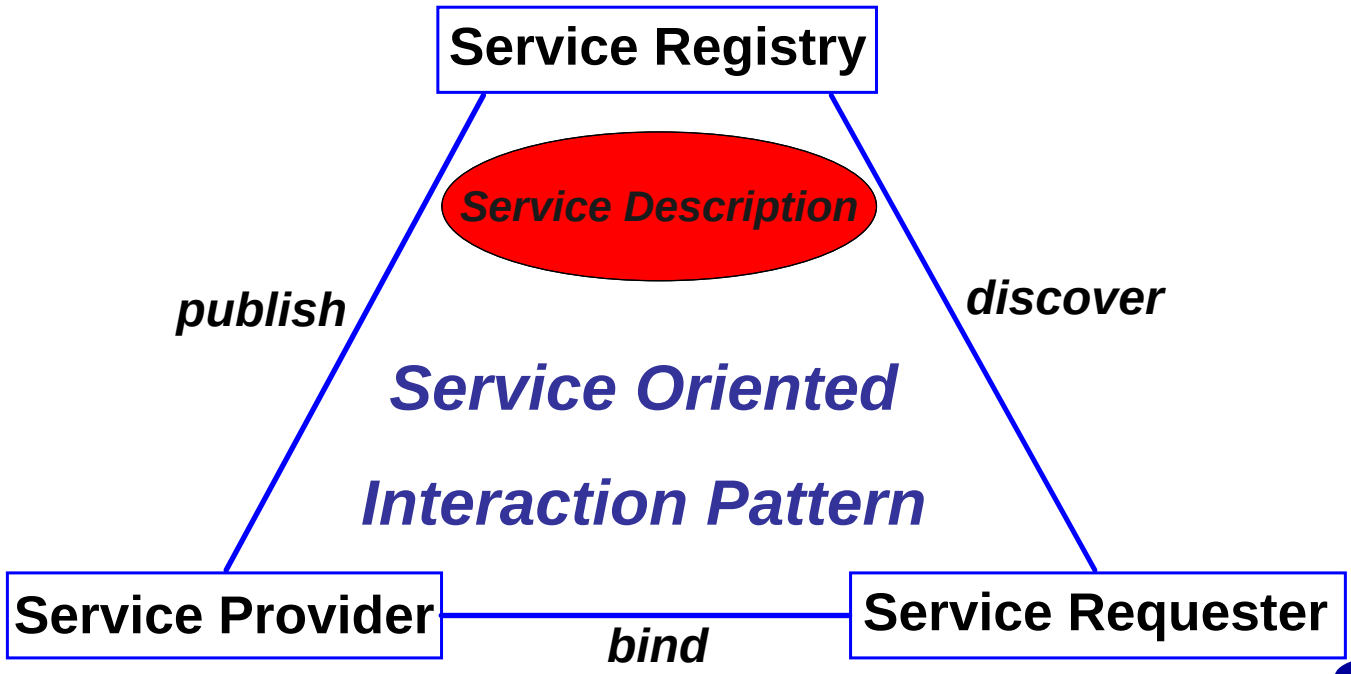
Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - … one ore more service provider references (if there exist) are returned …



Service Orientation (2/5)

- Support of dynamic discovery (*find-bind-execute* paradigm):
  - … after ***binding*** a service provider, a reference to the *"service object"* implementing the service functionalities is returned.



Service Orientation (3/5)

- Service requestor *is not tied* to a particular service provider:
  - service *providers may be substituted* whenever they *obey to the contract* imposed by the service description.

- Due to *dynamic availability*, a service may be removed from the registry at any time:
  - running application must be able to *releasing* (*incorporating*) *departing* (*arriving*) services.

Service Orientation (4/5)

- To exhibit with *dynamic availability:*
  - service platforms should provide *notification mechanisms* used to inform service requesters of the arrival or departure of the services.
  - others might use the concept of a *service lease*, which means that service availability is guaranteed only for a determined time period after which the lease must be renewed.

Service Orientation (5/5)

- *Service composition* is an abstract composition:
  - Since it is based *only on service descriptions*, it *concretizes* only at run-time.
- *Hierarchical service composition* is achieved when a service composition itself has a service description.
- Service composition can be realized through standard programming language, although a *flow-based approach* is favored in the domain of Web Services (e.g., processes such as BPEL).

Service-orientation principles!

- Service-oriented development platforms should provide infrastructures to support building applications according to *service-orientation principles*.
- Examples of other platforms include the CORBATrader, Jini, OSGi, IBM WebSphere, etc.

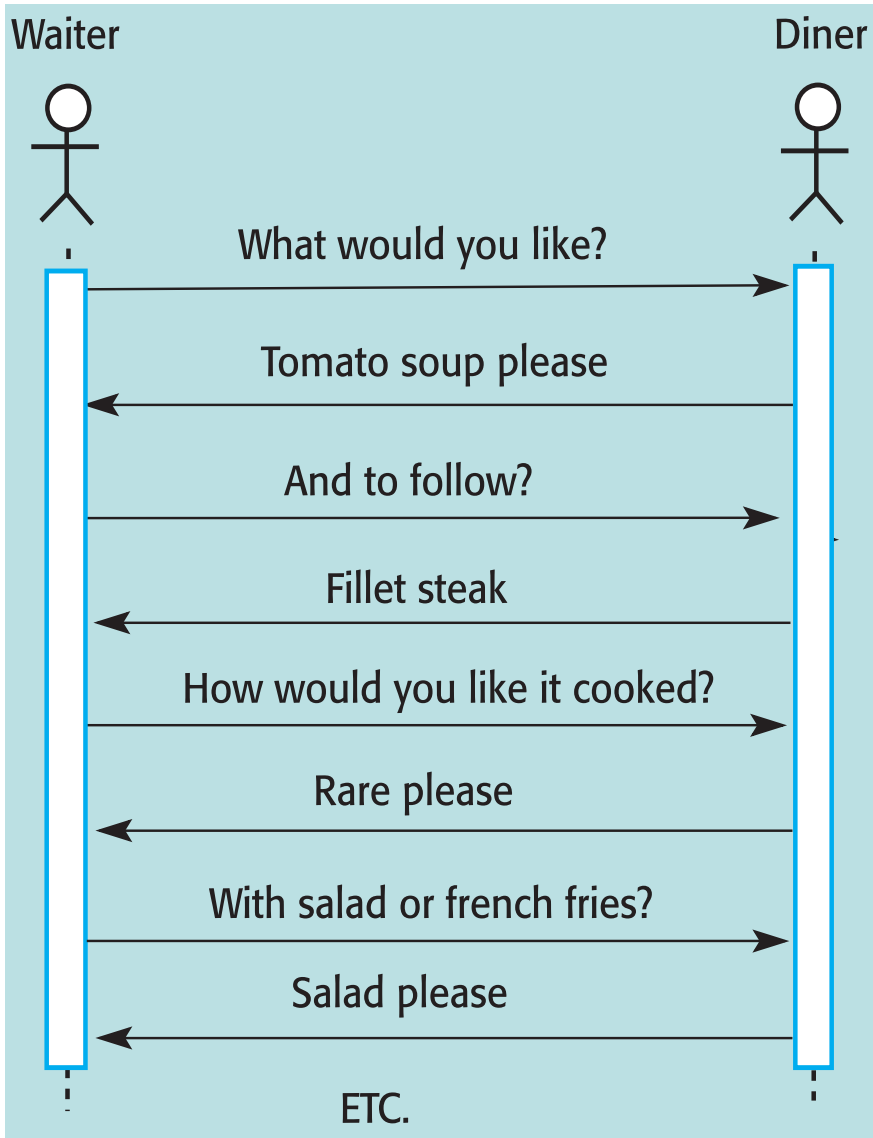
Service Orientation *VS* Component Orientation

(both promote reuse by separating building blocks from assembly)

- Emphasis on software service descriptions;
- Natural separation from description and implementation;
- Only service description is available during assembly;
- Integration of service provider is made prior or during execution;
- Better support for run-time discovery and substitution;
- Native dynamic availability;
- Abstract composition.

- Emphasis on component external-view;
- No evident separation between component external-view and implementation;
- Component physically available during assembly;
- Integration at assembly-time;
- (Typically) No support for run-time discovery and difficult substitution.
- (Typically) No dynamic availability;
- Structural composition.

Service Orientation *VS* Component Orientation
(Message Passing *VS* Method Calls – e.g., RPC/RMI)

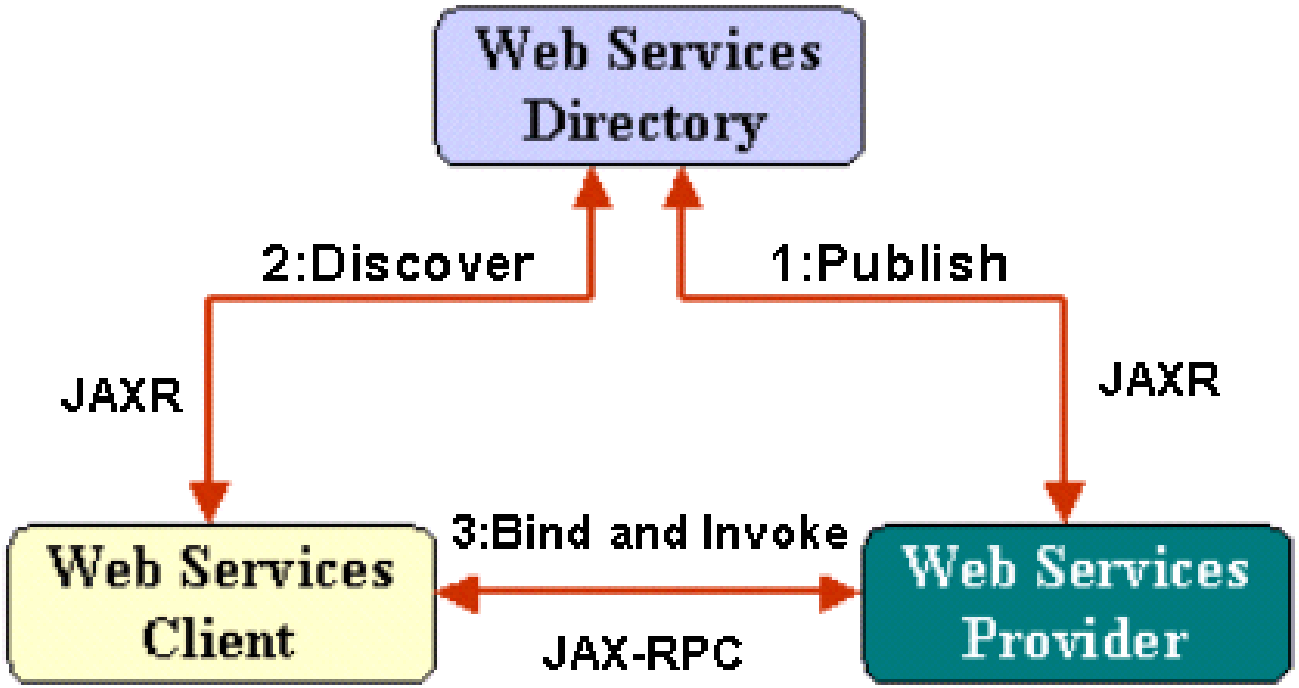


An order as an XML message

```
<starter>
     <dish name = “soup” type = “tomato” />
     <dish name = “soup” type = “fish” />
     <dish name = “pigeon salad” />
</starter>
<main course>
     <dish name = “steak” type = “sirloin”
cooking = “medium” />
     <dish name = “steak” type = “fillet”
cooking = “rare” />
     <dish name = “sea bass”>
</main>
<accompaniment>
     <dish name = “french fries” portions =
“2” />
     <dish name = “salad” portions = “1” />
</accompaniment>
```

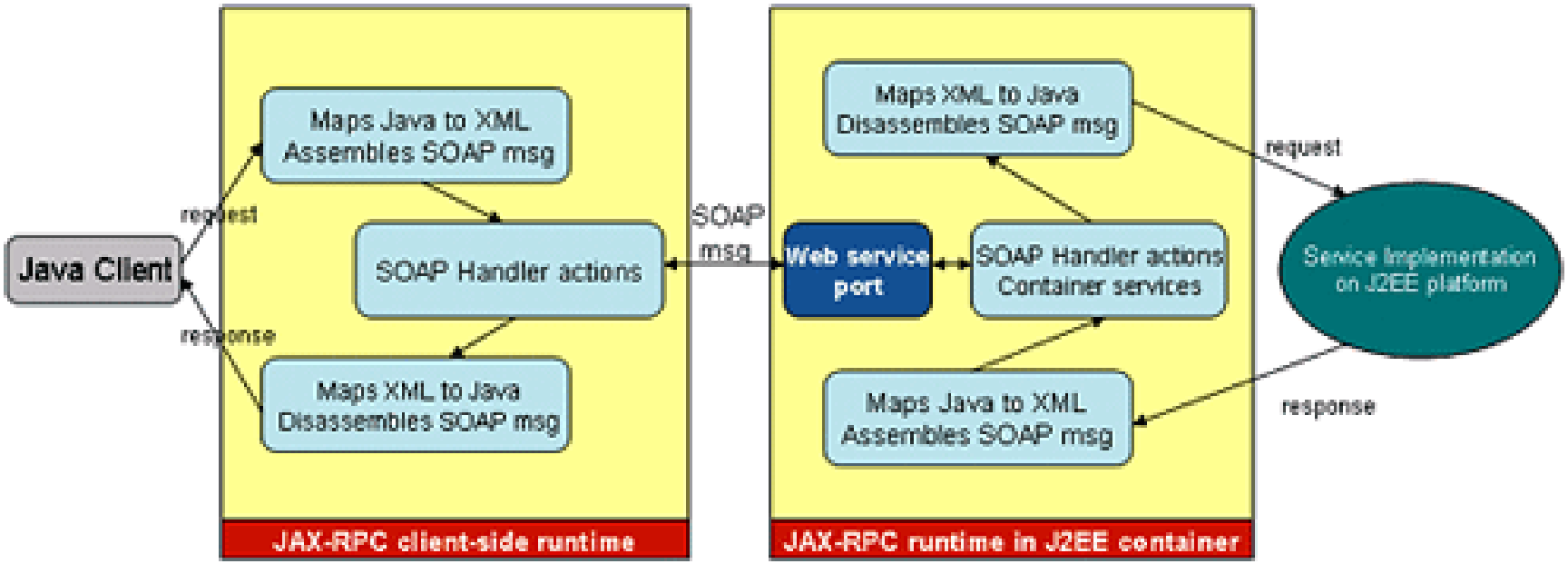
In practice

- JAXR and JAX-RPC APIs play a role in publishing, discovering, and using web services and thus realizing SOA

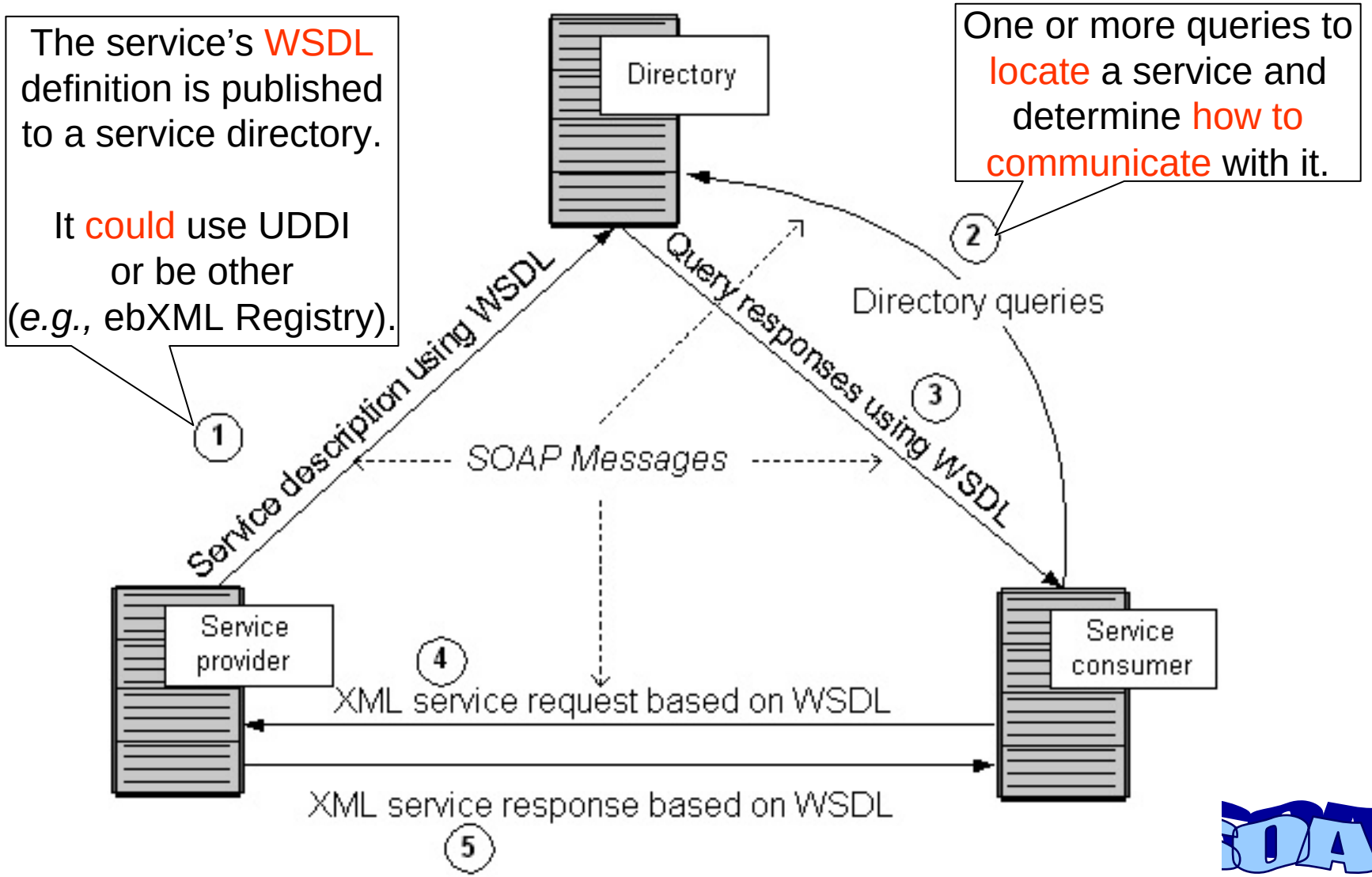


In practice

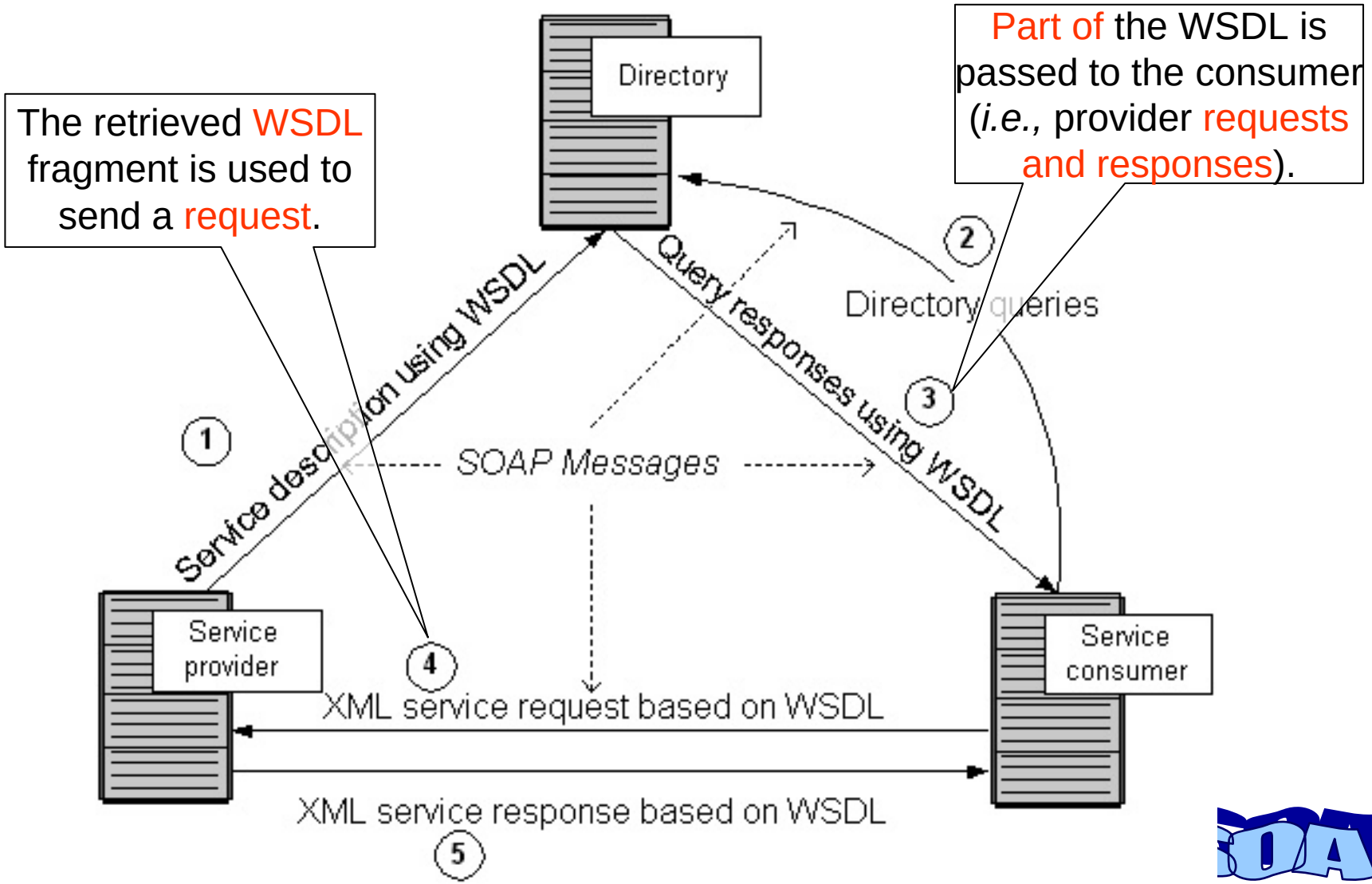
- All the details between the request and the response happen behind the scene



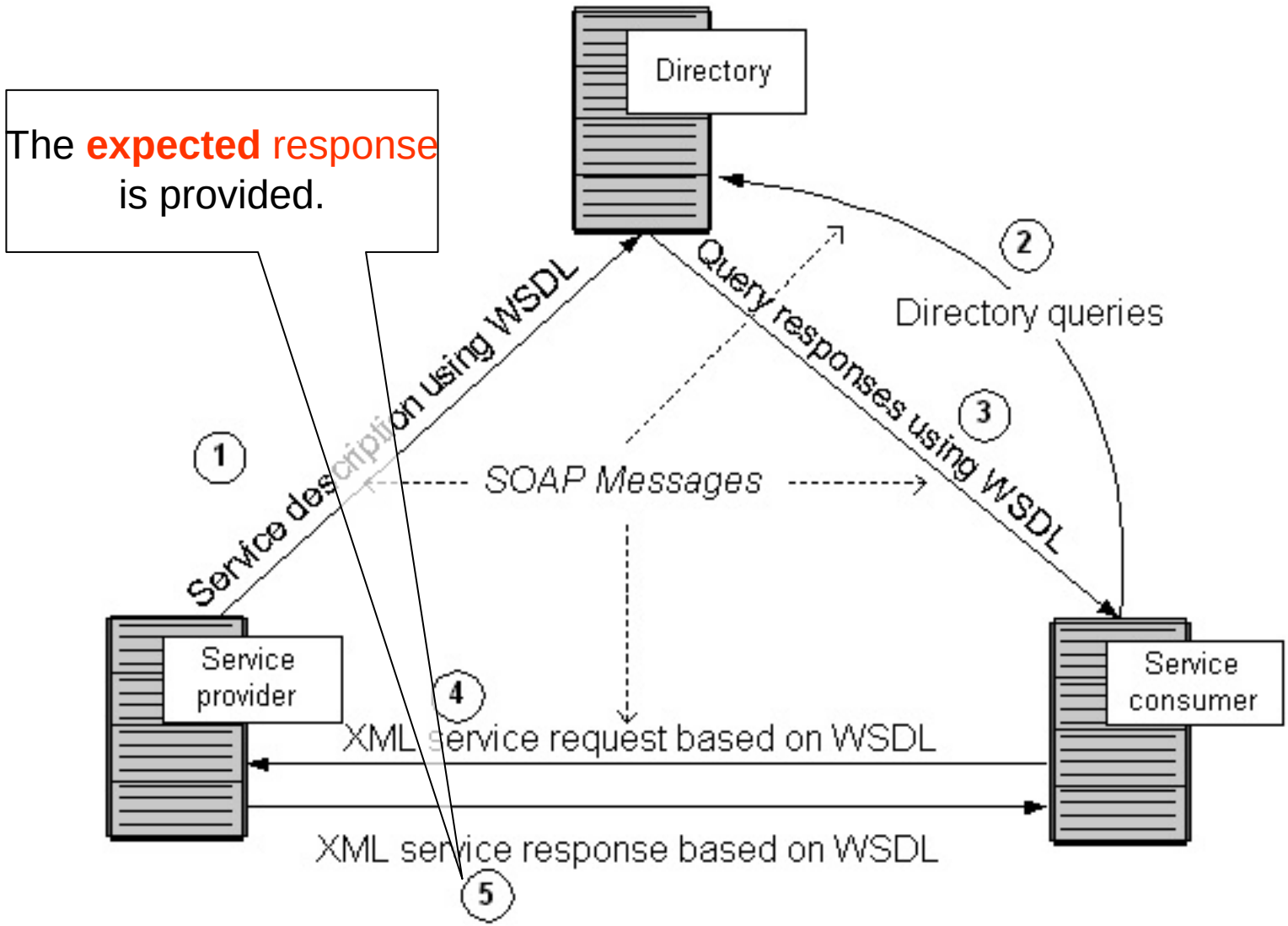
Web Service Interaction Pattern (In practice)

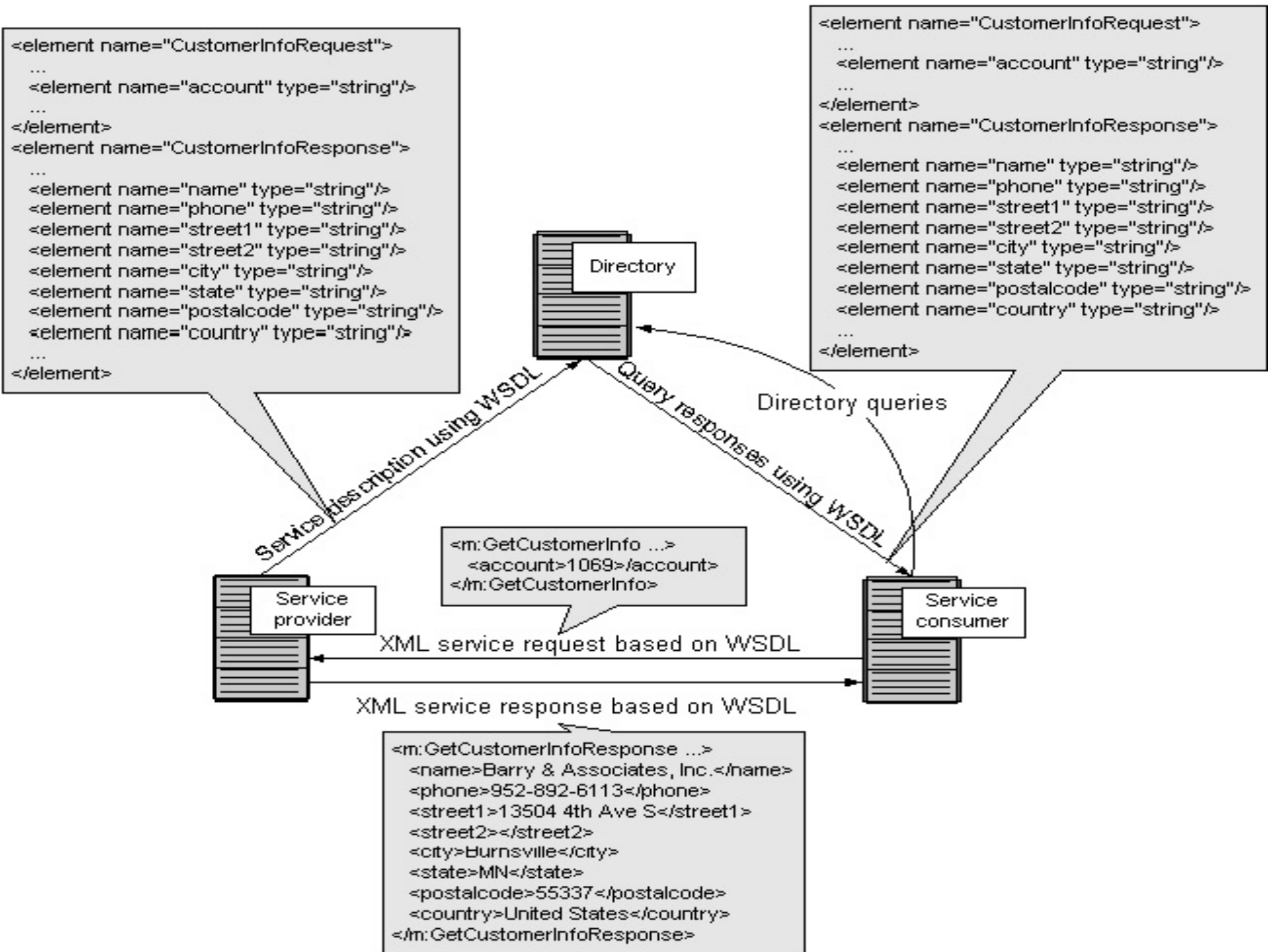


Web Service Interaction Pattern (In practice)



Web Service Interaction Pattern (In practice)




<element name="CustomerInfoRequest">
...
<element name="account" type="string"/>
...
</element>
<element name="CustomerInfoResponse">
...
<element name="name" type="string"/>
<element name="phone" type="string"/>
<element name="street1" type="string"/>
<element name="street2" type="string"/>
<element name="city" type="string"/>
<element name="state" type="string"/>
<element name="postalcode" type="string"/>
<element name="country" type="string"/>
...
</element>
Directory
Service description using WSDL
Query responses using WSDL
Directory queries
Service provider
Service consumer
<m:GetCustomerInfo ...>
<account>1069</account>
</m:GetCustomerInfo>
XML service request based on WSDL
XML service response based on WSDL
<m:GetCustomerInfoResponse ...>
<name>Barry & Associates, Inc.</name>
<phone>952-892-6113</phone>
<street1>13504 4th Ave S</street1>
<street2></street2>
<city>Burnsville</city>
<state>MN</state>
<postalcode>55337</postalcode>
<country>United States</country>
</m:GetCustomerInfoResponse>

Universal Description Discovery and Integration (UDDI)

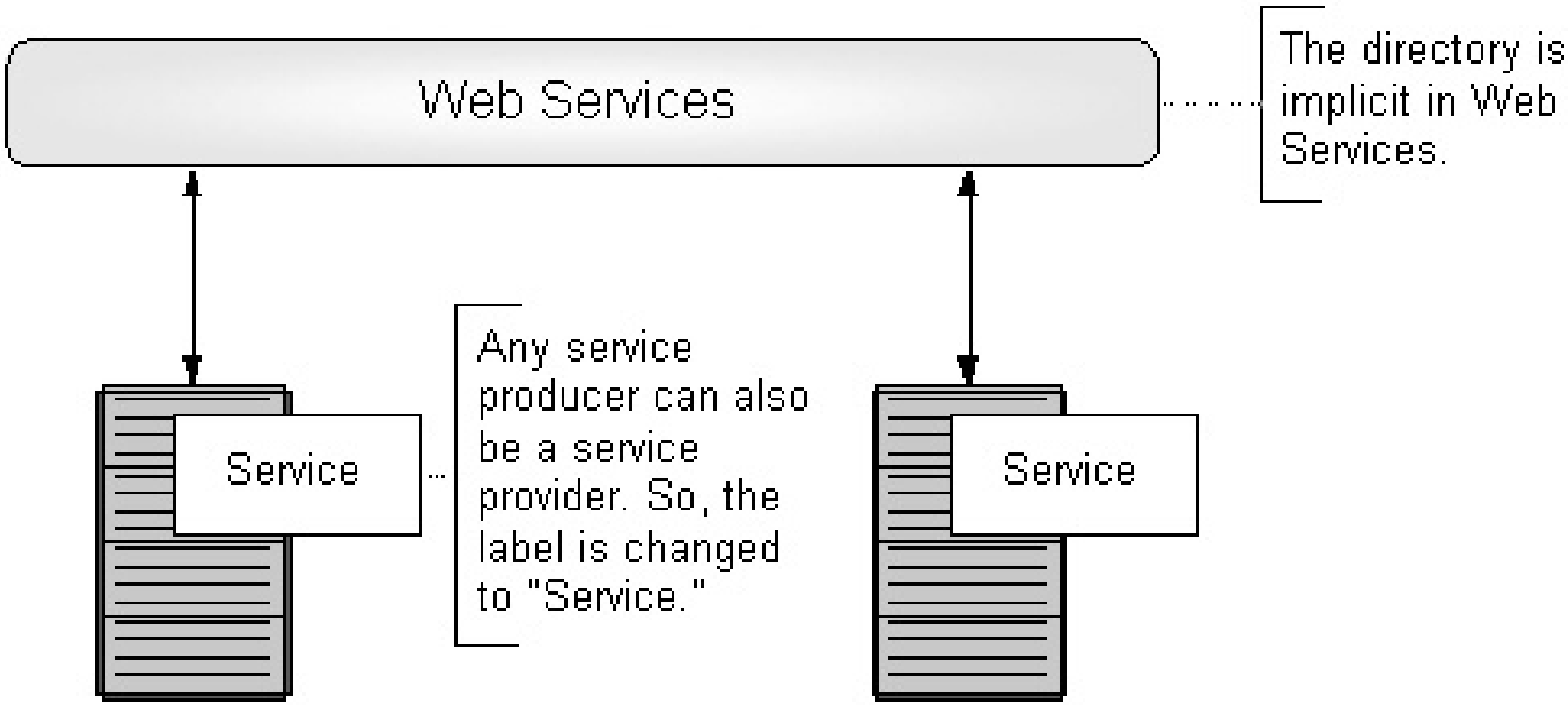
- A *"phone book"* of Web Service (WS) providers.
- Defines a set of services supporting the description and discovery of
  - WS providers;
  - their available WS;
  - technical interfaces to access them.
- UDDI was first developed by UDDI.org and then transferred to OASIS .

UDDI structure

- White pages:
  - business identifiers, *e.g.*, company name, address, phone/fax numbers.
- Yellow pages:
  - business data organized by classifications, *e.g.*, by the NAICS (North American Industry Classification System).
    - *"It i a production-oriented conceptual framework, groups establishments into industries based on the activity in which they are primarily engaged".*
    - *"Establishments using similar raw material inputs, similar capital equipment, and similar labor are classified in the same industry. In other words, establishments that do similar things in similar ways are classified together". ...it is a kind of ontology.*
- Green pages:
  - business processes, *e.g.*, operating platform, supported programs, purchasing methods, shipping and billing requirements.

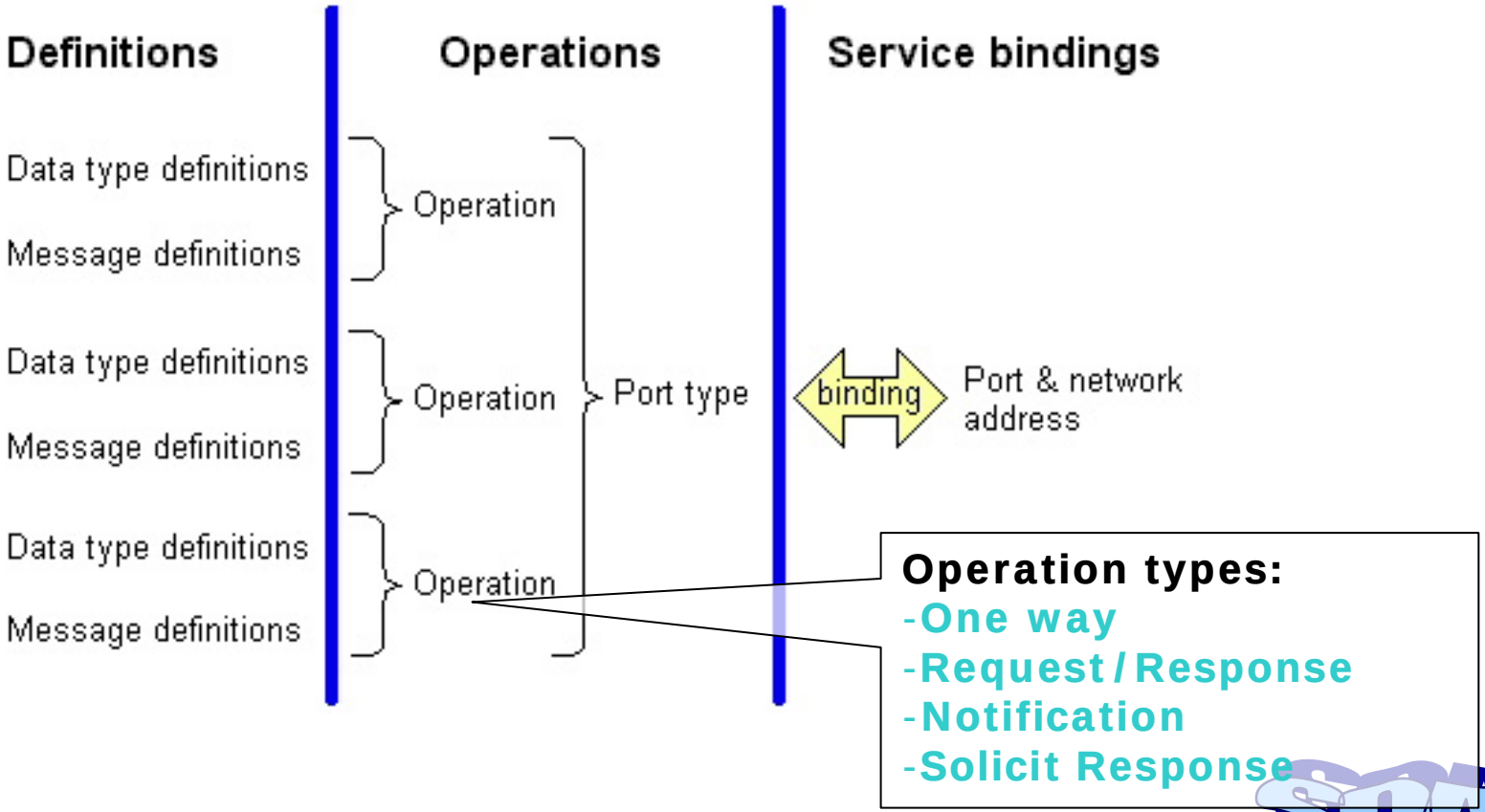
Simplified WS architecture: the Bus style

- Used by other middleware solutions
  - first SOAs in the past were based on: DCOM, ORBs, CORBA, .NET, JMS, and WebSphere MQ (IBM).



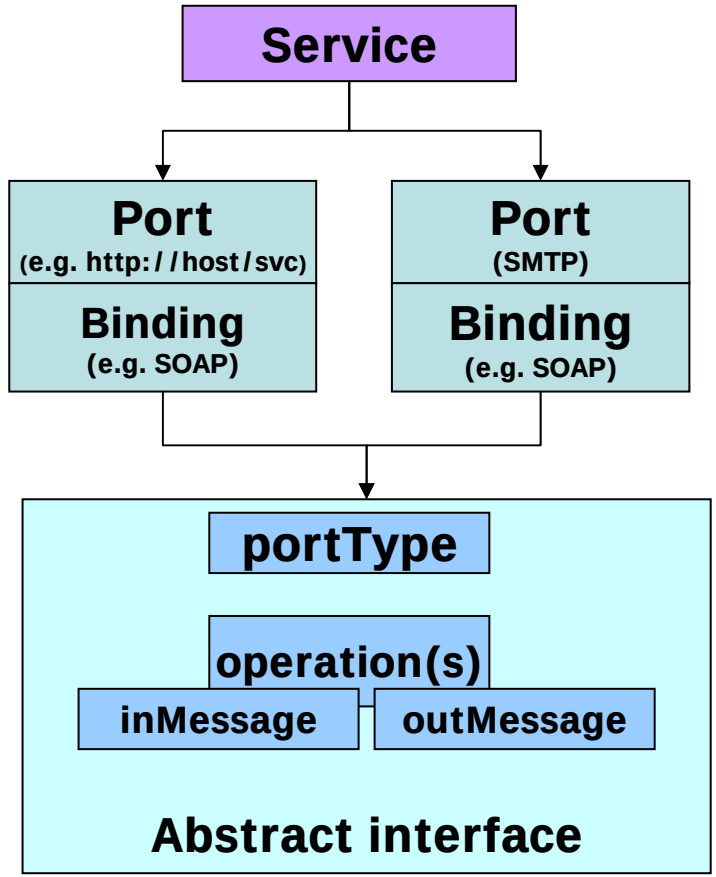
WSDL structure (1/2)

- Extensible by `<any>...</any>`



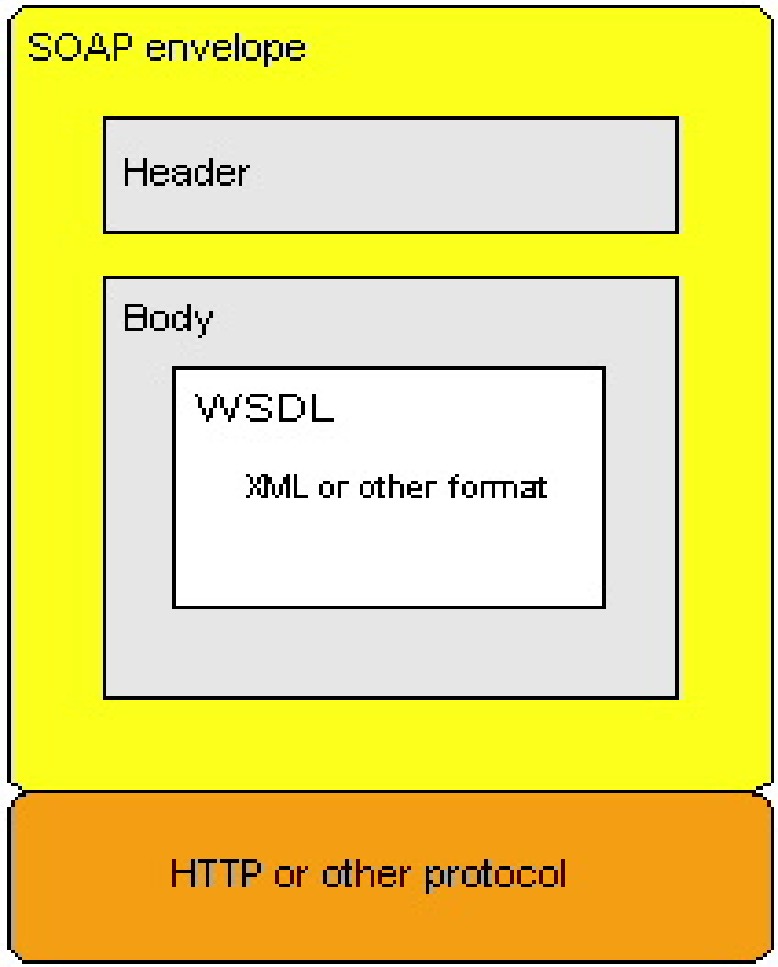
WSDL structure (2/2)

- PortType:
  - abstract definition of a service (set of operations);
- Multiple bindings per portType:
  - how to access it;
  - SOAP, JMS, direct call;
- Ports:
  - where to access it.



SOAP messages

- Header:
  - authentication information;
  - data encoding;
  - messages processing method.
- Body:
  - WSDL definition of a service;
  - WSDL definition of a request;
  - WSDL definition of a response.

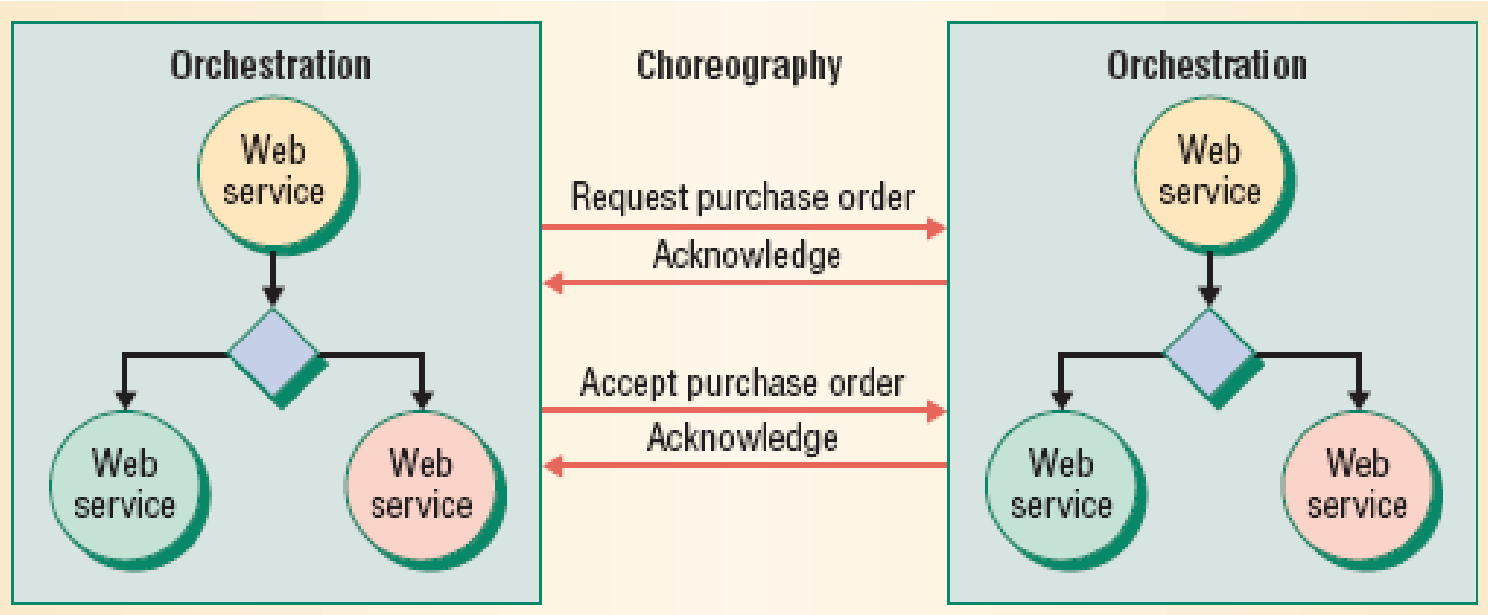


Orchestration and Choreography

- Building complex Web Services requires combining other Web Services.
- Choreography operates on top of orchestration.
- They are complementary: orchestration is used to “define” choreography.

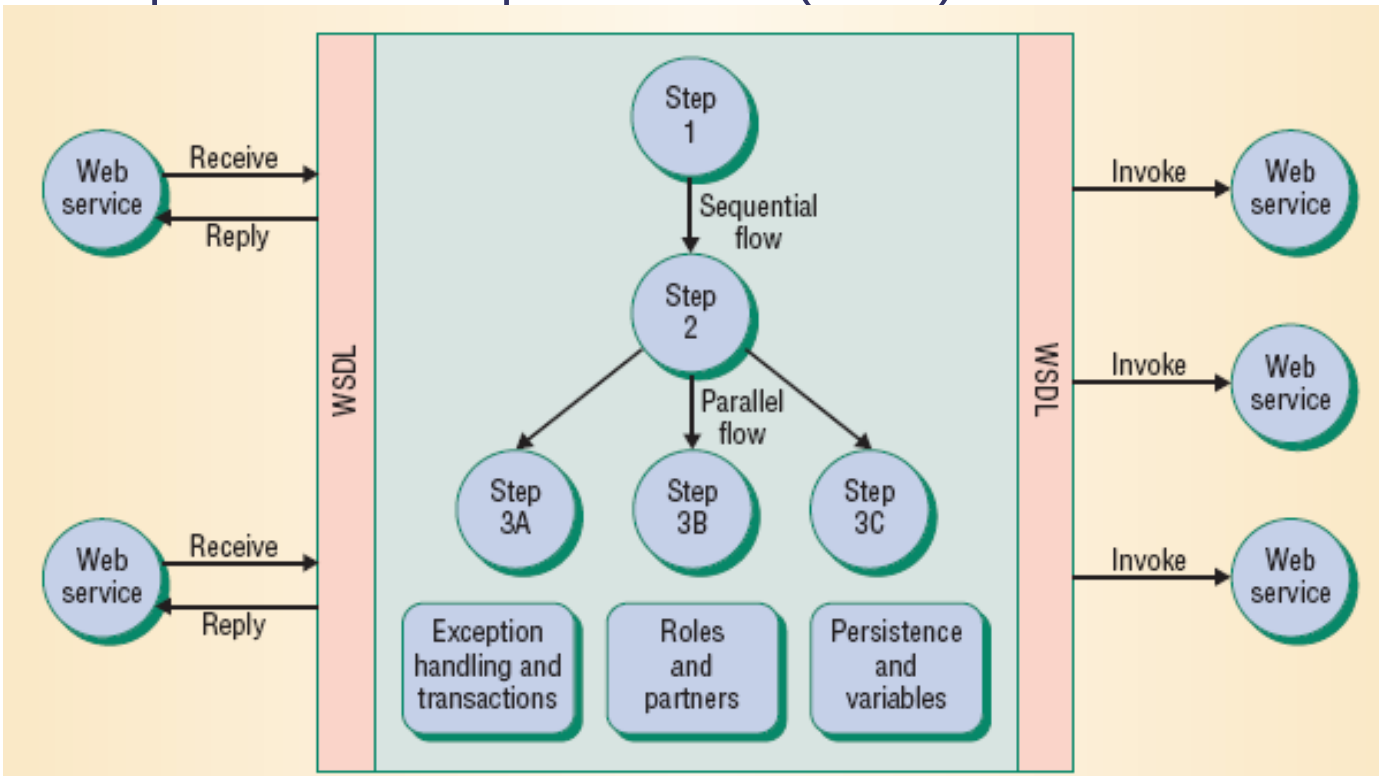
Orchestration versus choreography

- Orchestration refers to an executable process (internal and external message level).
- Choreography tracks the (external) message sequences between services.



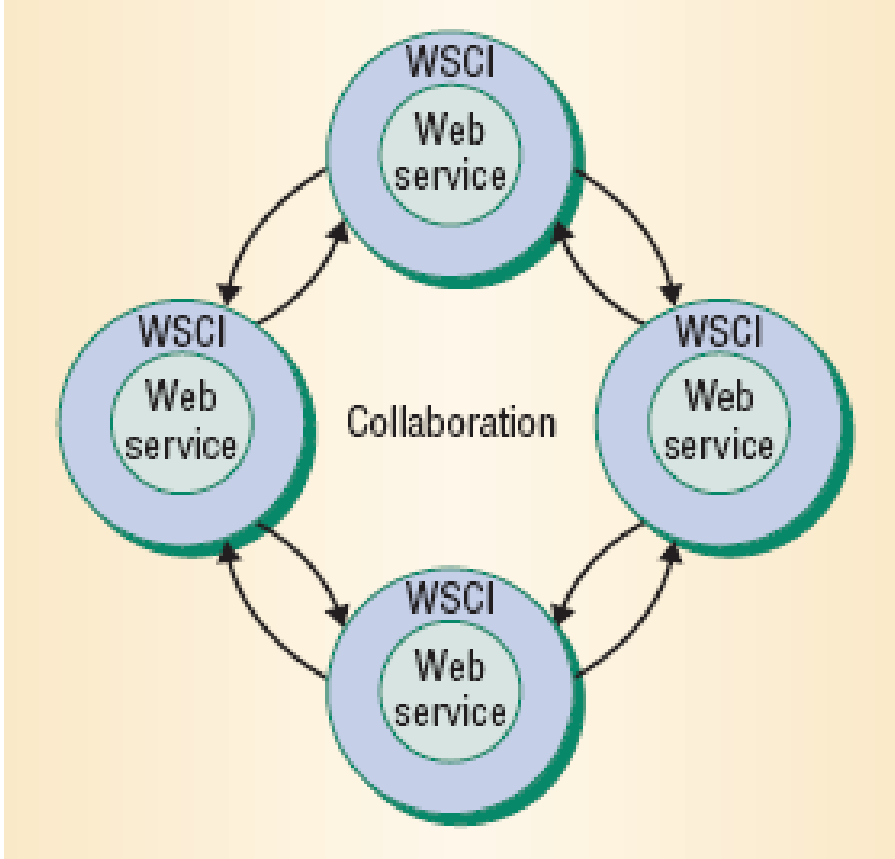
Orchestration

- Process flow specification languages, *e.g.*, BPEL4WS (Business Process Execution Language):
  - structured activities to manage the overall process;
  - basic activities concerning interactions with external services;
  - process flow exposed as WS (WSDL).



Choreography

- WS Collaboration Interface (WSCI) specification languages:
  - observable interaction behavior between WSs (and not the definition of an executable business process).



Service + Component Orientation

Service as reusable components

- *Recalling that:*
  - *"CBSE promotes the construction of software applications as composition of reusable building "blocks" called components based on some specific standard model" [Szyperski].*

- In the future, service might be a natural development of components if the component model will be a *unique set of widely accepted standards*.

- A (Web) Service can be defined as:
  - *"A service is a loosely-coupled, reusable software component that encapsulates discrete functionality which may be distributed and programmatically accessed.*
  - *A web service is a service that is accessed using standard Internet and XML-based protocols" [Sommerville].*

Service + Component Orientation

- A service-oriented component model should introduces concepts of service orientation into a component model.
- This combination facilitates *dynamism* in a component model:
  1. Service description "as" a **contract.**
  2. Components **contractually implement** service descriptions;
  3. A service-oriented **interaction pattern** is used to contractually resolve service dependencies;
  4. Abstract composition **in terms of contracts** (late, non-explicit binding among components via services);
  5. Contracts are the basis for **substitutability**;

(1) Service as a contract

- *A service is provided functionality: it is a reusable operation or set of operations.*

- *A contract defines a service's characteristics for discovery, access and composition. This is achieved by describing some combination of the service's syntax, behavior, semantics and QoS.*

- *An important syntactical aspect of a contract is that it explicitly declares service's "contractual dependencies" on other services to enable structural composition.*

(2) Components contractually implement service descriptions

- *By implementing a contract the component provides the service functionalities and obeys its constraints.*
- *In addition to the service "contractual dependencies" specified in the contract, a component may also declare additional "implementation-specific" dependencies (once again to enables structural composition) BUT …*
- *… service description are the sole means of interaction among component instances.*

(3) Service-oriented interaction pattern to resolve service dependencies

- Service description "implemented" by component instances are **published into a service registry**.

- The **service registry** is used to dynamically discover services at run-time for resolving service's *"contractual dependencies"*.

(4) Composition in terms of contracts

- An **abstract composition is a set of contracts** that are used to select concrete components to instantiate.
- **Explicit bindings are not necessary**, since they are inferred at run time from the service “contractual dependencies” declared in the constituent contracts.
- **Composition become concrete at run time** as component that provide the constituent contracts are dynamically discovered and instantiated into the composition.
- **Composition is a continuous run-time activity** that responds to the availability of services.

(5) Contracts as basis for substitutability

- It is **not necessary to select particular components** for a composition, since compositions are defined in terms of contracts that, in turn, form the basis for substitutability.

- In a composition, any component that implements a given contract **can be substituted** with any alternative component that implements the same contract.

To summarize

- **Dynamic availability and substitutability** are strengthened by two levels of dependencies:
  - **contractual**;
  - **implementation specific.**
- In component orientation, **only implementation dependencies exist**, which hinders substitutability.
- In service orientation, according to SOA principles, **service dependencies are not declares as part of the contract**, which eliminates the possibility of structural composition.
- **Dynamic availability** is also enabled because explicit bindings are not necessary, since they are inferred from the contracts.
- Finally, all **"service dependencies" are resolved at run time** using the service-orientation interaction pattern and this interaction pattern forms the basis of a **continual composition life cycle**.

Challenges in creating a service-oriented component model

- Ambiguity:
  - Multiple candidate services might exist suitable for solving a given service dependency. What is the "best"? Physically near? Inexpensive? Secure? Some other metric?
  - A solution for limiting ambiguity might be a rich discovery protocol which use semantics and behavior, in addition to syntax to reduce the number of candidates *(next slide)*.
- Dynamic Availability:
  - Continuous deployment/un-deployment due to context-changes → context-awareness;
  - Continuous listening for service/component departure or arrival;
  - Possible adaptation in response to non available perfectly fitting services/components →alternatives must be searched and eventually adapted.
  - …

Towards rich discovery protocols

For instance:

- *WSDL are primarily concerned with the syntactical description of functionality and interface :*
  - *Semantics and extra-functional characteristics (e.g., performance, dependability) are not defined.*
  - *The consumer/developer is in charge of understanding WHAT the service actually does.*
  - *Meaningful names and documentation helps here, but there is still the scope for misunderstanding and misuse.*

- *Adaptive Service Grid (ASG) semantically rich description by functional and non-functional requirements:*
  - *Service description by the ASG language are then translated into a deployable EJB component.*

SOA characteristics: modularity

- **Modular Decomposability:**
  - break of an application into many smaller module.
  - each module is responsible for a single, distinct functionality.
  - it is sometimes referred to as "top-down design", in which the bigger problems are iteratively decomposed into smaller problems.
  - Service designers *should* identify the smallest unit of software that can be reused in different contexts

SOA characteristics: modularity

- **Modular Composability:**
  - Modularly composing a service refers to the production of software services that may be freely combined as a whole with other services to produce new systems
  - It is sometimes referred to as bottom-updesign.
  - Service designers *should* create services sufficiently independent to be reused in entirely different applications from the ones for which they were originally intended.

SOA characteristics: modularity

- **Modular Understandability:**
  - A service *should* be described in such a way that a person is able to understand its offered functionalities (and other specifications) without having any knowledge of other services.
  - e.g., If the behavioral specification of a service is tied to many other (not strictly related) specifications, developer/consumer will have hard time to making a decision and reuse the service.

SOA characteristics: modularity

- **Modular Continuity:**
  - the impact of changes in one service should not require changes in other services or in the consumers of the service.
  - An service description that does not sufficiently hide the implementation details of the service creates a domino effect when changes are needed:
    - other service may have assumption on this implementation details.
- Every service *must* hide information about its internal design.

SOA characteristics: modularity

- **Modular Protection:**
  - Therefore, a service *must* designed so to ensure that, during its consumption, faults do not cascade from the service to other services or consumers:
    - e.g., faults in the operation of a service must not impact the operation of a client, or other service, or the state of their internal data, otherwise the contract with service consumers will be broken.

SOA characteristics

- **Interoperability:**

  - SOA stresses **interoperability**:
    - the ability of systems to use different platforms and languages to communicate with each other.

  - Protocol and data format should be standardized:
    - for instance *Java API for XML-based Remore Procedure Call* (JAX-RPC) and *Java API for XML-based Messaging* (JAXM) map Java data types to SOAP and vice-versa.

SOA characteristics

- Loose Coupling:
  - The binding from the service requester to the service provider should loosely couple the service:
    - i.e., the service requester has no knowledge of the technical details of the provider's implementation, such as the programming language, deployment platform.
  - Implementation on each side of the conversation can change without impacting the other, provided that the message schema stays the same.
  - Legacy code, such as COBOL, can be replaced with new Java code without having any impact on the service requester.

First service classification

service functional classification

- **Context adaptive and intelligent user services**:
  - These are “ambient intelligent” type of services providing users the ability to access the services they need, anywhere from any device.
- **Information services**:
  - services that provide (personalized) information, for instance by comparing, classifying, or otherwise adding value to separate information sources.
- **Intermediary services**:
  - services that help in finding appropriate services, for instance search services.
- **Location-based services (LBS)**:
  - family of service that depend on the knowledge of the geographic location of mobile stations.

Second service classification (1/2)

service technical protocol
or business classification

- **Web Services**:
  - support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (WSDL).
  - Interact by WS description using SOAP-message (usually conveyed using HTTP with an XML serialization and other Web-related standards.

- **P2P services**:
  - are services “p2p” networks, meaning sharing files and content between computers via direct interaction between users on the network, facilitated by a virtual name space (VNS).
  - A VNS associates user-created names with the “physical” IP.
  - User do not need to know the location of other users.

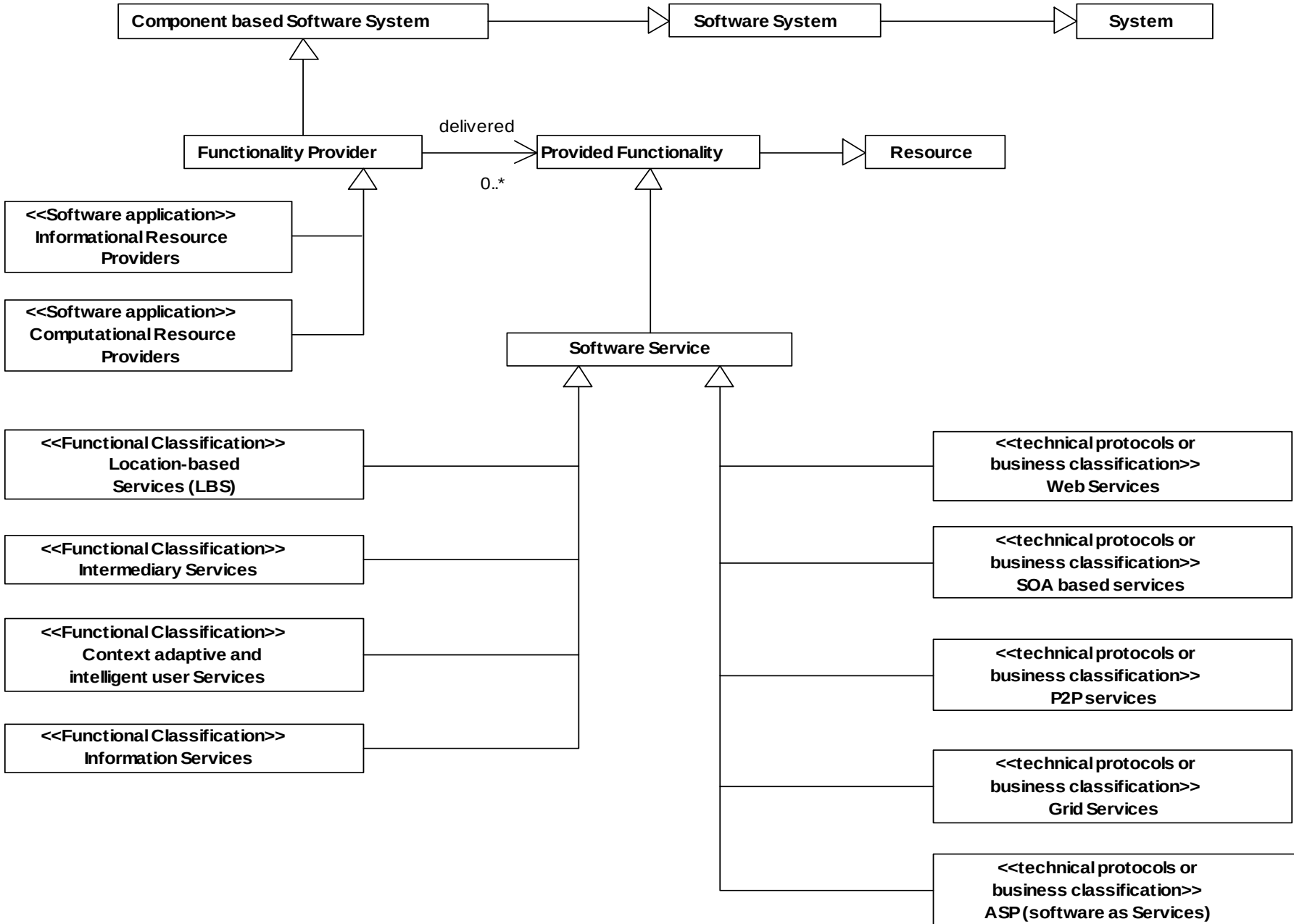
Second service classification (2/2)

- **Grid services**:
  - Grid is a type of parallel and distributed computation based on aggregation of geographically distributed "autonomous" resources dynamically at runtime. Grid exploits ability to share and aggregate distributed computational capabilities and deliver them as service.

- **ASP, software as a service**:
  - ASP is an abbreviation of Application Service Provider, a third-party entity that manages and distributes software-based services to customers across the network.
  - In essence, ASP's are a way for companies to outsource some their need of information technology.

- Many others service classifications exist in the Literature.

Service classification



Challenges

- SOA-based applications are distributed multi-tier applications that have presentation, business logic, and persistence layers.
- Services are the building blocks of SOA applications.
- While any functionality can be made into a service, the challenge is:
  - to define a service (and its description) that is at the right level of abstraction “meeting the business view”.
  - Services should provide coarse-grained functionality.

Challenges

- *Service description:* How should we describe services in order to have real world abstract services correctly represented in a description?
- *QoS:* Which QoS measurements should we take into account in order to deliver services that gain the users confidence that the services will operate as they expect?
- *Monitors:* How should we monitor services in order to be sure that a service provider really supply the service as described and with the declared QoS?

Moreover

- How should we *preserve* the user expected Quality of Service when moving across different infrastructures?
- How should we tackle run time discovery and services composition to have efficient *dynamic adaptation.*
- How should we address *service adaptation to the context of use* spanning from the Provider context of use to the Consumer context of use?

OSGi overview

- Open Services Gateway (initiative):
  - home services gateways (HSGs) as initial target;
  - (home area) networked devices;
  - an HSG acts as a mediator between the end user of the devices and the providers that want to provide services for the devices.
- Typical examples:
  - home security;
  - home health care monitoring.

OSGi Service Platform

- Adding it to a networked device adds the capability to manage the life cycle of the software components in the device from anywhere in the network.
- Software components can be installed, updated, or removed on the fly without having to disrupt the operation of the device.
- Software components are applications or libraries that can dynamically discover or use other components.

OSGi framework

- It is service oriented.
- Four layers:
  - Security
    - java 2 security;
  - Module
    - modularization model for java (sharing or hiding packages between bundles);
  - Life cycle
    - dynamically install, update, or uninstall bundles;
  - service registry
    - support for the run-time selection of specific implementations with respect to specific needs or vendors.

Summarizing (1/3)

- SOA and web services are two different things, but web services are the preferred standards-based way to *"realize"* SOA.
- SOA is an architectural style for building software applications that use services available in a network such as the web.
- SOA promotes loose coupling between software components so that they can be reused. Applications in SOA are built based on services.
- A service is an implementation of a well-defined business functionality, and such services can then be consumed by clients in different applications or business processes.

Summarizing (2/3)

- SOA allows for the reuse of existing assets where new services can be created from an existing IT infrastructure of systems.
- In other words, it enables businesses to leverage existing investments by allowing them to reuse existing applications, and promises interoperability between heterogeneous applications and technologies.

Summarizing (3/3)

SOA provides a level of flexibility that wasn't possible before in the sense that:

- Services “are software components” with well-defined interfaces that are implementation-independent. An important aspect of SOA is the separation of the service interface (the **WHAT**) from its implementation (the **HOW**). Such services are consumed by clients that are not concerned with how these services will execute their requests
- Services are self-contained (perform predetermined tasks) and loosely coupled (for independence)
- Services can be dynamically discovered
- Composite services can be built from aggregates of other services
- SOA uses the **find-bind-execute** paradigm

Epilogo

- This slides are a selection of various information concerning SOA retrieved from the Web, Literature and from my own experience.

References

- **[1]** C. M. Anne-Marie Sassen. The service engineering area. *An overview of its current state and a vision of its future*, July 2005.
- **[2]** Humberto Cervantes and Richard S. Hall. *Autonomous adaptation to dynamic availability using a service-oriented component model*, ICSE 2004. **(CONSIGLIATO – Serv. + Comp. viene da qui)**
- **[3]** W3C Group. Web services architecture (WSA)
- **[4]** SUN. Service Oriented Architecture. *http://www.theserverside.com* **(CONSIGLIATO – search SOA)**
- **[5]** *ASG, IST FP6 project nr 004617.*
- **[6]** SeCSE project: Service Centric System Engineering.
- **[7]** IBM, http://www-128.ibm.com/developerworks/webservices/newto/ **(CONSIGLIATO)**
- **[8]** PLASTIC project, Providing Lightweight and Adaptable Service technology for pervasive Information and Communication.
- **[9]** PLASTIC IST STREP Project. Deliverable D1.2: Formal description of the PLASTIC conceptual model and of its relationship with the PLASTIC platform toolset.
- **[10]** Software Engineering, IAN Sommerville, 8 Edition **(CONSIGLIATO)**


That's ALL Folks
THANK YOU