

# Service-oriented architecture and Web services

# Needs for software integration

- **Business point of view**
  - constantly changing requirements
  - efficient utilization of information systems, e.g., cost-effectiveness
    - security in information exchange
  - etc.
- **Software engineering point of view**
  - constantly changing requirements
  - reuse aspect
  - needs for information exchange
  - efficiency
  - flexibility
  - etc.

## Integration architectures, service-oriented architectures...

- Integration architecture
  - defines and describes a way to integrate
    - software systems
    - components or parts of software
    - information needed and used by software systems
  - not necessarily a "service point of view"
- Service-oriented architecture
  - an example of integration architecture
  - Dr. Hao He, W3C Web Services Architecture Working Group:  
*Service Oriented Architecture is an architectural style whose goal is to achieve loose coupling among interactive software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer.*
  - uses integration techniques
    - service coordination (possibly dynamically) from other services

## ... and Web services

- One way to implement SOA
  - WSDL, SOAP
  - RESTful services
- In practice – and in research – these terms get mixed every now and then
  - people talk about integration architectures and service-oriented architectures as they were the same thing
  - people talk about service-oriented architectures and Web services as they were the same thing

## SOA - requirements and principles from software engineering point of view

- As loose dependencies among services as possible
  - aim at minimizing the dependencies, esp. the artificial ones
  - only real dependencies
  - the "unity of information" should be possible to be defined inside one service, not among several services
    - independent services
- Late configuration and service binding
  - services are possibly searched at run-time
  - connections between services are formed dynamically at run-time

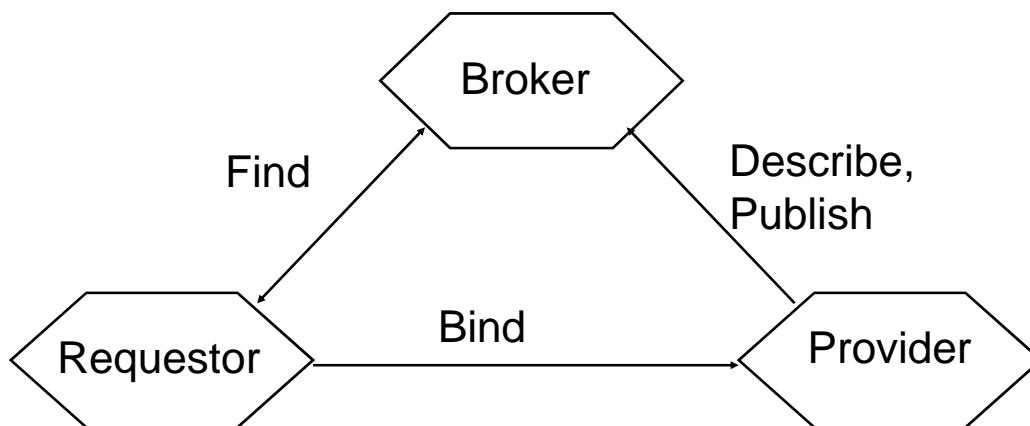
## SOA - requirements and principles from software engineering point of view

- Aiming at long living services
  - potentially increases the amount of users...if the service is shown to be useful
  - better possibilities for the users to develop ways to manage error situations, interoperability problems etc.
- Independence from implementation methods and techniques
  - interfaces are important, not how the services have been implemented (c.f. distributed systems)
- High level of abstraction
  - parties offer services at relatively high level of abstraction

## SOA - requirements and principles from software engineering point of view

- **Autonomy**
  - services are independent and their management is decentralized
  - in addition to establishing connections, the independent parties may also negotiate and make agreements on the forms of communication... ..dynamically
- **Agile management of requirements**
  - ability to react on changing and new requirements ("a meta requirement")
- **Statelessness**
  - If a service needs to retain its state for a longer period of time, its availability to other clients may be impeded
  - Managing state information can also hinder scalability
- **Service composability**
  - it should be possible to coordinate and assemble services to form composite services

## Service-Oriented Architecture (SOA)



Dr. Hao He, W3C Web Services Architecture Working Group:

*Service Oriented Architecture is an architectural style whose goal is to achieve loose coupling among interactive software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer.*

## Provider

- Provides services
  - e.g., on-line hotel reservation
- Describes its service (in WSDL) based on the specification prepared by a service Broker
  - *describe* operation
- Registers its service with Broker
  - *publish* operation
- Requestor is be able to invoke a registered operation (with *bind* operation)

## Requestor

- Looks for services (more precisely: asks a Broker to look for services) and uses them
- May also create a new service by integrating services provided by Providers
  - e.g., a travel agency using on-line hotel reservation and flight booking services
- Scenario:
  - 1) invokes Broker to get services of interest (*find* operation)
  - 2) selects a service/services and then invokes the corresponding Provider(s) (*bind* operation)
- Becomes a Provider if it registers itself with Broker
  - hierarchy of services

## Broker

- Registers service descriptions submitted by Providers (e.g., using UDDI)
  - *publish* operation
- Looks for services (i.e., executes a search) when asked
  - *find* operation
  - e.g., a travel search site for finding hotels according to given preferences
- Defines the service description format and an API for registering and searching services
- It is actually also a Provider

## What's new in SOA then?

- A pure software architecture point of view
  - In principle, SOA is not very interesting and it does not seem to offer that much new... (c.f. component technologies)
  - However, ...
    - loose coupling can be implemented in an interesting way using e.g. Web service technologies
      - an agreement on Web service standards...more or less: WSDL, SOAP, and possibly others (e.g. UDDI)
    - not just for RPC
      - document-style communication
    - one goal is to distribute not only the services but also their management
      - decentralized applications
    - SOA can be considered "a step to the next abstraction level"
      - components -> services

## What's new in SOA then?

- A big claim about SOA: it brings the software engineering and business points of views closer to each other!
  - not that clear before (c.f. OO and component technologies)
  - systems consist of service that have their own business processes
  - business processes and their comprehension as well as understanding the requirements are essential in the development of services
  - care and attention is needed when talking about the concepts and terms: e.g. *service* and *architecture* from software engineering and business points of views

## SOA and Web services

- Web services offer one way to implement SOA
- SOA and Web services aim at automatically solving problems related to interoperability, configuration etc.
  - an agreement on the standards partly enables this
    - challenges: a whole range of "standards"
  - possible problems:
    - nobody really knows the code, what if there is a need to understand the code....
    - genericity/flexibility vs. automation
- A lot of tool support for building Web services and client applications
  - varying features
  - support for Web service languages and recommendations, and their different versions varies

# Layers of Web services

Application services

Collaboration services  
(orchestration, choreographies)

SOAP and its extensions (for realibility,  
transactions etc.), WSDL, UDDI

Transport  
(HTTP, SMTP, FTP, JMS, IIOP, etc.)

Utility services  
(security, transactions, QoS, etc.)

## Web service formats

- **Web Service Description Language (WSDL)**
  - a format for describing a Web service interface, data, message types, interactions patterns, and protocol mappings
- **Simple Object Access Protocol (SOAP)**
  - a message format for invoking a Web service
  - binds the WSDL description

# REpresentational State Transfer (REST)

- Proposed by Roy Fielding
  - In his PhD dissertation "Architectural styles and the design of network-based software architectures", University of California, Irvine, USA, 2000.
- ReST is an architectural style (originally for distributed hypermedia systems) that aims at retaining interoperability among systems that may evolve independently. In order to achieve this goal, ReST defines a set of interface constraints.
- Later, ReST has been proposed as an alternative view to loosely-coupled services in general.

# ReST – Representational State Transfer

- Representation
  - Bytes that represent a *resource*
    - A resource can be anything we are interested in and that can be identified (by a URI)
    - One resource can have many representations
- State
  - A resource has a state at the server
  - A client application has a state
- Transfer
  - Client applications transfer states *to* the server in order to update server's resource state
  - Client applications get states *from* the server to update its own state

## ReST characteristics and constraints

- Application state and functionality are divided into *resources*
  - ReST is resource-oriented
- Every resource is uniquely addressable using a *universal syntax* for use in *hypermedia links*
- All resources share a *uniform interface* for the transfer of state between client and resource, which consists of standard way to address of resources and fixed set of operations
  - Resources are identified by URIs
  - Typical ReST applications use HTTP: standard methods (get, post, put, delete) and standard respond codes

## WWW – and example of ReSTful design

- The Web consists of HTTP, content types (e.g. HTML), and other internet technologies, such as DNS (Domain Name System)
- HTML can include javascript and applets to support “code-on-demand” principle
- HTTP has a uniform interface for accessing resources, which consists of URIs, methods, status codes, headers, and content distinguished by MIME type.

## ReST challenges

- **Security**
  - Rely on HTTP authentication framework (...only)
  - No standard for message-level security, c.f. Web services
  - WS-Security might have some reusable solutions
- General and comprehensive service description standard (c.f. WSDL)...is there already: WADL
- Service compositions, choreographies/orchestrations
  - Would be, I believe, as relevant as in case of Web services