www.mas-ict.ch

Hes·so
Haute École Spécialisée de Suisse occidentale

Java EE & Web Services

Distributed Architecture
Service-Oriented Architecture

Omar ABOU KHALED

Stefano CARRINO

Julien Tscherrig

University of Applied Sciences of Western Switzerland
EIA-FR, Bd de Pérolles 80 - CP 32, CH-1705 Fribourg
omar.aboukhaled@hefr.ch |Tél: +41 26 429 65 89
stefano.carrino@hefr.ch| Tél: +41 26 429 67 48
http://humantech.eia-fr.ch | Fax: +41 26 429 66 00

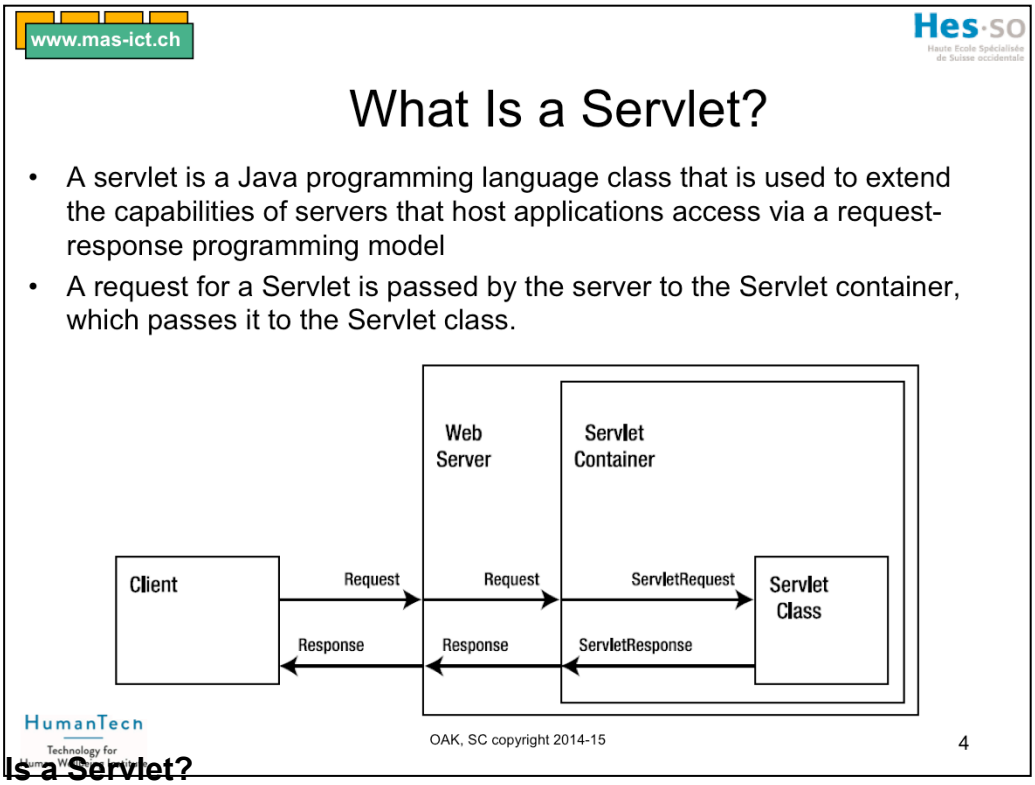
HumanTech
Technology for Human Wellbeing Institute

Plan

- Day 1
  - Introduction to architecture evolution
    - 2 tiers, 3 tiers, distributed, etc.
  - Introduction to Java EE
    - Servlet and JSP
- Day 2 & 3
  - Java EE: EJB, JPA and JSF
  - Web Services: UDDI, WSDL, SOAP
  - An Overview of Java Web Services
  - Service-Oriented Architecture (SOA)
  - RESTful web services

Overview

- Java Sever Faces (JSF)
- Service Oriented Architecture
  - SOAP Web Services



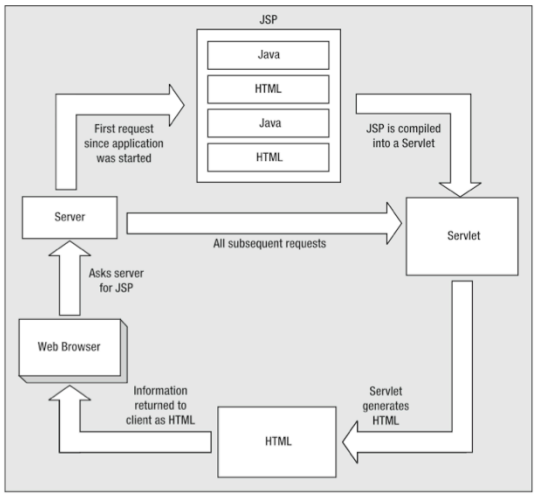
What Is a Servlet?

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications access via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers.

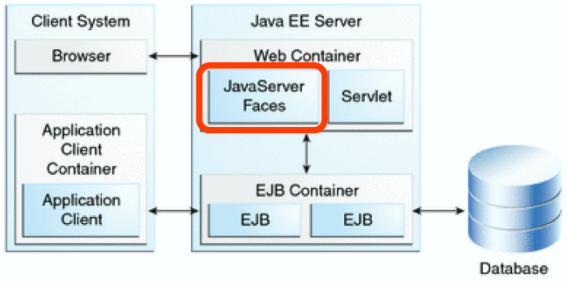
When a client (usually, but not necessarily, a web browser) makes a request to the server, and the server determines that the request is for a Servlet resource, it passes the request to the Servlet container. The container is the program responsible for loading, initiating, calling, and releasing Servlet instances. The Servlet container takes the HTTP request; parses its request URI, the headers, and the body; and stores all of that data inside an object that implements the javax.servlet.ServletRequest interface. It also creates an instance of an object that implements javax.servlet.ServletResponse. The response object encapsulates the response back to the client. The container then calls a method of the Servlet class, passing the request and response objects. The Servlet processes the request and sends a response back to the client.

JavaServer Pages Technology

- The main features of JSP technology are as follows:
  - A language for developing JSP pages, which are text-based documents that describe how to process a request and construct a response
  - An expression language for accessing server-side objects
  - Mechanisms for defining extensions to the JSP language

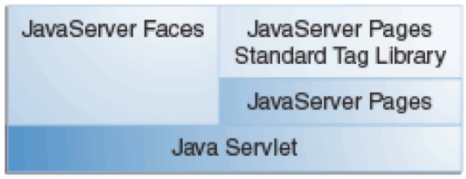


When a web server receives a request for a JSP, it passes the request to the JSP container (not shown in figure). If the JSP page has not been translated, the container translates the JSP into a Java Servlet source file, and then compiles the source file into a class. The Servlet class is loaded and the request is passed to the class. The Servlet processes the request and returns the result to the client. All subsequent requests are routed directly to the Servlet class, without the need to translate or compile again.



JavaServer Faces

JSF



A server side user interface component framework for Java technology-based web applications.

Architecture

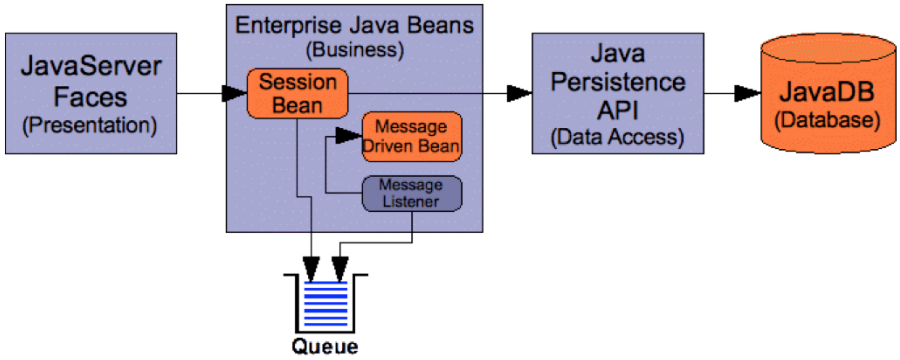
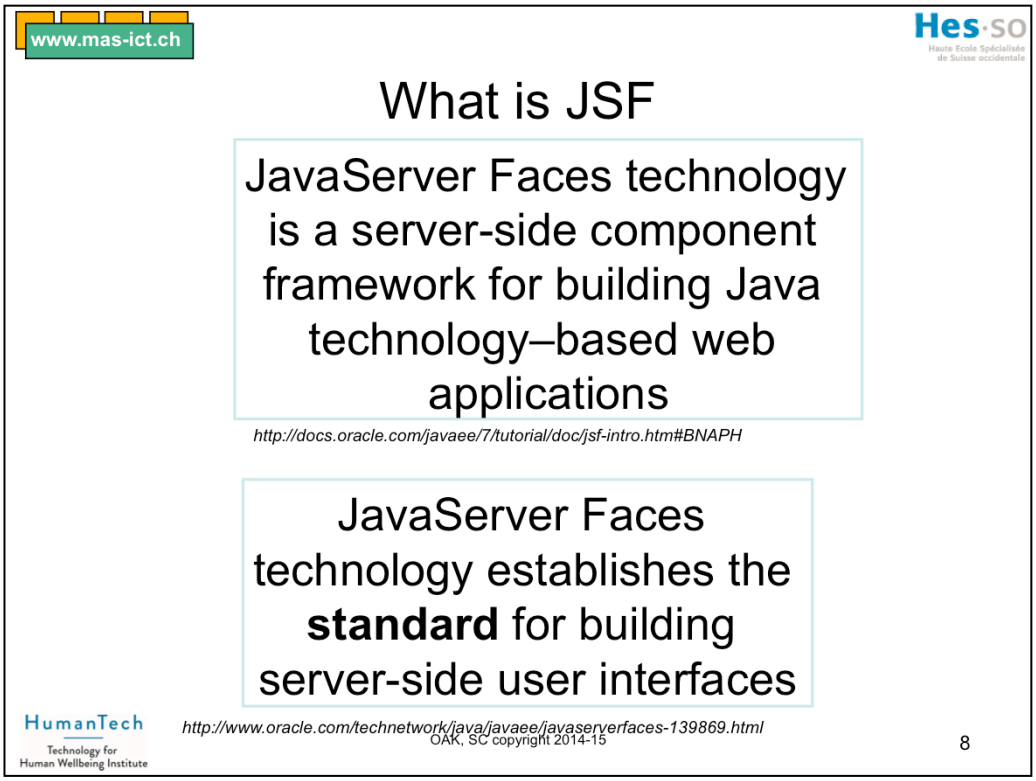


Image source: http://miageprojet2.unice.fr/index.php?title=Intranet_de_Michel_Buffa/Design_Patterns_Java_EE_6/Design_patterns_Java_EE_6_with_JSF_%2F%2F_EJB_%2F%2F_JPA

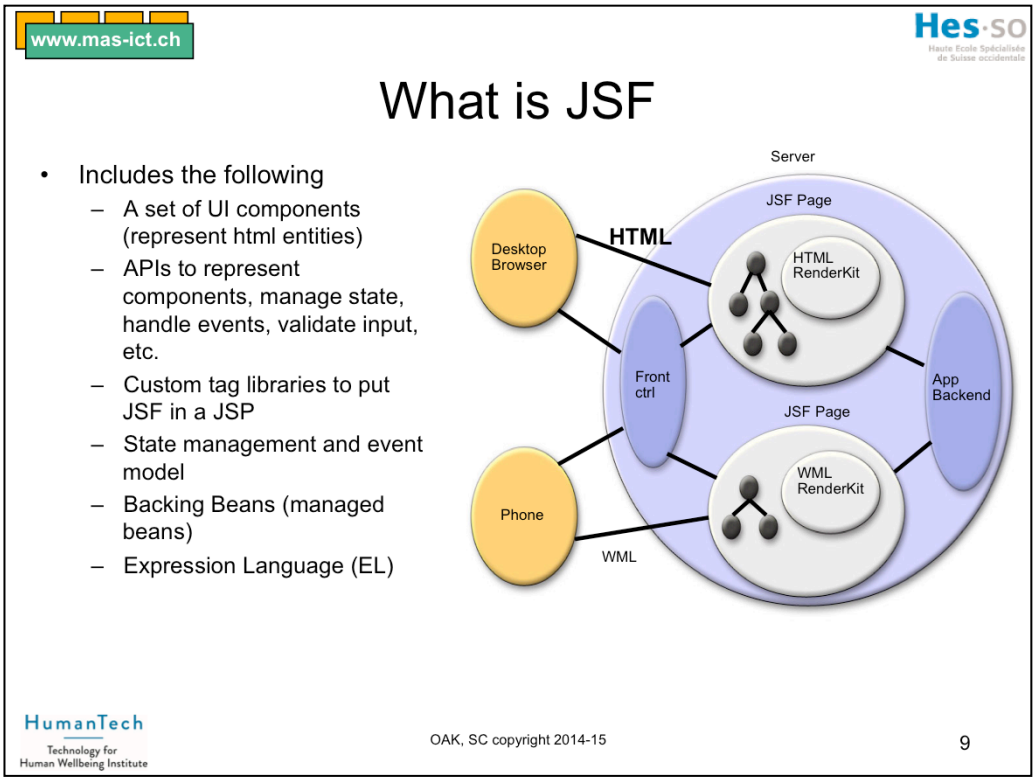


JSF Vs JSP Vs Facelets

JSF is a standardized Java framework for web UIs based on an MVC pattern

JSPs are a (much older) standard for generating web pages from templates - these can be used as the View in a JSF application, but also separately from JSF.

Facelets are an alternative view technology based on pure XML templates (no scriptlets) which was introduced with Version 2 of the JSF standard. They can only be used in a JSF application.



JavaServer Faces (JSF)

This picture shows JSF architecture in a somewhat simplified manner. Just like any other MVC-based architecture, JSF architecture has its own Front controller called FacesServlet. A JSF page is made of a tree of UI components. These UI components can be associated with backend model objects called backing beans. These backing beans handle application logic (or sometimes called business logic) handling.

JSF is a relatively new technology that attempts to provide a robust, rich user interface for web applications. JSF is used in conjunction with Servlets and JSPs. JSF provides a component-based API for building user interfaces. The components in JSF are user interface components that can be easily put together to create a server-side user interface. The JSF technology also makes it easy to connect the user interface components to application data sources, and to connect client-generated events to event handlers on the server.

The JSF components handle all the complexity of managing the user interface, leaving the developer free to concentrate on business logic. The flexibility comes from the fact the user interface components do not directly generate any specific presentation code. Creating the client presentation code is the job of custom renderers. With the correct renderer, the same user interface components could be used to generate presentation code for any arbitrary device. Thus, if the client's device changed, you would simply configure your system to use a renderer for the new client, without needing to change any of the JSF code.

Why ?

JSP and Servlet: No built-in UI component model

Struts (I am not saying you should not use Struts): No built-in UI component model, No built-in event model for UI components, No built-in state management for UI components, No built-in support of multiple renderers, Not a standard (despite its popularity)

Apache Struts: http://en.wikipedia.org/wiki/Struts

Apache Struts is an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to encourage developers to adopt a model-view-controller (MVC) architecture.

What you can do with JSF?

- Create a web page.
- Drop components onto a web page by adding component tags.
- Bind components on a page to server-side data.
- Wire component-generated events to server-side application code.
- Save and restore application state beyond the life of server requests.
- Reuse and extend components through customization.

A set of web pages in which components are laid out.

A set of tags to add components to the web page.

A set of managed beans, which are lightweight, container-managed objects (POJOs). In a JavaServer Faces application, managed beans serve as backing beans, which define properties and functions for UI components on a page.

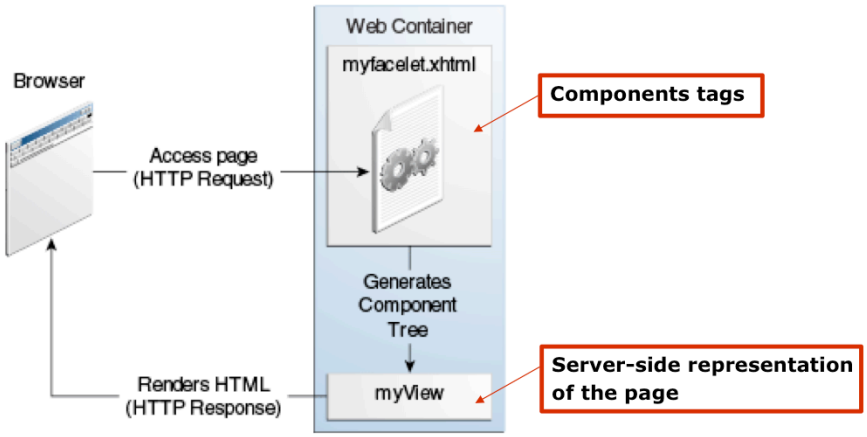
A web deployment descriptor (web.xml file).

Optionally, one or more application configuration resource files, such as a faces-config.xml file, which can be used to define page navigation rules and configure beans and other custom objects, such as custom components.

Optionally, a set of custom objects, which can include custom components, validators, converters, or listeners, created by the application developer.

Optionally, a set of custom tags for representing custom objects on the page.

Responding to a Client Request for a JavaServer Faces Page



The web page, myfacelet.xhtml, is built using JavaServer Faces component tags. Component tags are used to add components to the view (represented by myView in the diagram), which is the server-side representation of the page. In addition to components, the web page can also reference objects, such as the following:

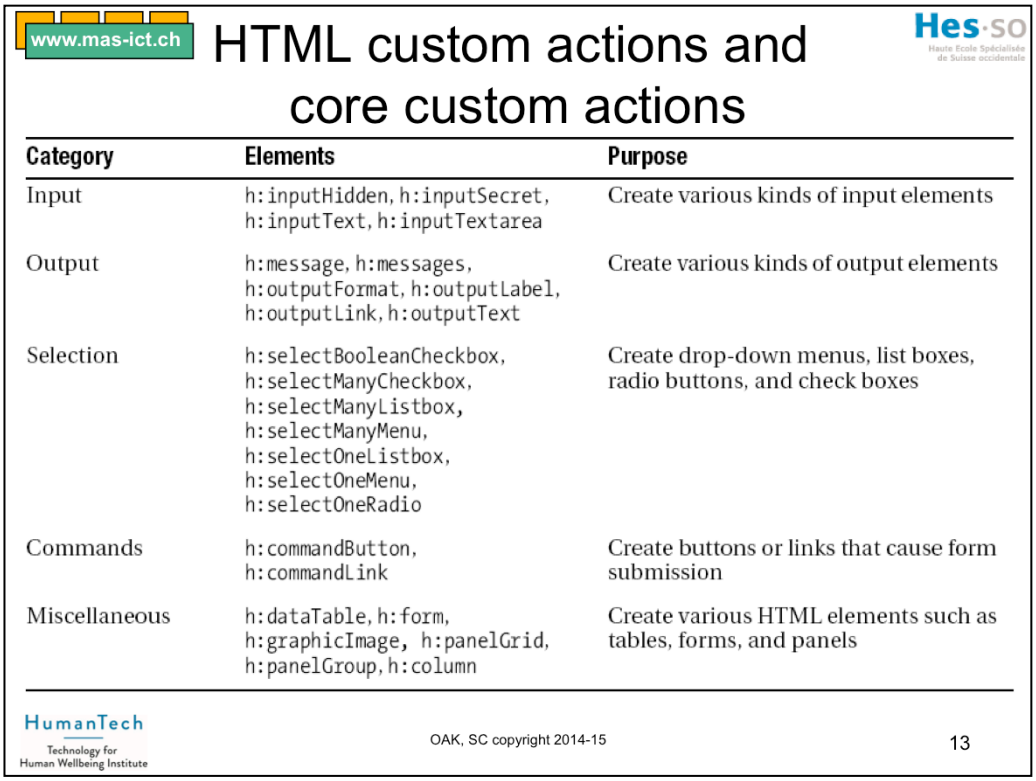
- Any event listeners, validators, and converters that are registered on the components
- The JavaBeans components that capture the data and process the application-specific functionality of the components

On request from the client, the view is rendered as a response. Rendering is the process whereby, based on the server-side view, the web container generates output, such as HTML or XHTML, that can be read by the client, such as a browser.

JSF Page Example

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html lang="en"
      xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelets Hello Greeting</title>
    </h:head>
    <h:body>
        <h:form>
            <h:graphicImage url="#{resource['images:duke.waving.gif']}"
                            alt="Duke waving his hand"/>
            <h2>Hello, my name is Duke. What's yours?</h2>
            <h:inputText id="username"
                         title="My name is: "
                         value="#{hello.name}"
                         required="true"
                         requiredMessage="Error: A name is required."
                         maxlength="25" />
            <p></p>
            <h:commandButton id="submit" value="Submit" action="response">
            </h:commandButton>
            <h:commandButton id="reset" value="Reset" type="reset">
            </h:commandButton>
        </h:form>
        ...
    </h:body>
</html>
```

```
package javaeetutorial.hello1;

import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named
@RequestScoped
public class Hello {

    private String name;

    public Hello() {
    }

    public String getName() {
        return name;
    }

    public void setName(String user_name) {
        this.name = user_name;
    }
}
```



Category	Elements	Purpose
Input	h:inputHidden, h:inputSecret, h:inputText, h:inputTextarea	Create various kinds of input elements
Output	h:message, h:messages, h:outputFormat, h:outputLabel, h:outputLink, h:outputText	Create various kinds of output elements
Selection	h:selectBooleanCheckbox, h:selectManyCheckbox, h:selectManyListbox, h:selectManyMenu, h:selectOneListbox, h:selectOneMenu, h:selectOneRadio	Create drop-down menus, list boxes, radio buttons, and check boxes
Commands	h:commandButton, h:commandLink	Create buttons or links that cause form submission
Miscellaneous	h:dataTable, h:form, h:graphicImage, h:panelGrid, h:panelGroup, h:column	Create various HTML elements such as tables, forms, and panels

Components of JSF

What is a component?: In JSF, a component is a group of classes that together provide a reusable piece of web-based user interface code. A component is made up of three classes that work together. They are: Render class, UIComponent subclass, JSP custom action class.

The HTML custom actions fall into five categories or elements: input, output, selection, commands, and miscellaneous. To use the HTML and Core custom tag libraries in a JSP page, you must include the taglib directives in the page. Example:

Taglib directives

<%@ taglib uri="http://java.sun.com/jsf/html/" prefix="h" %>

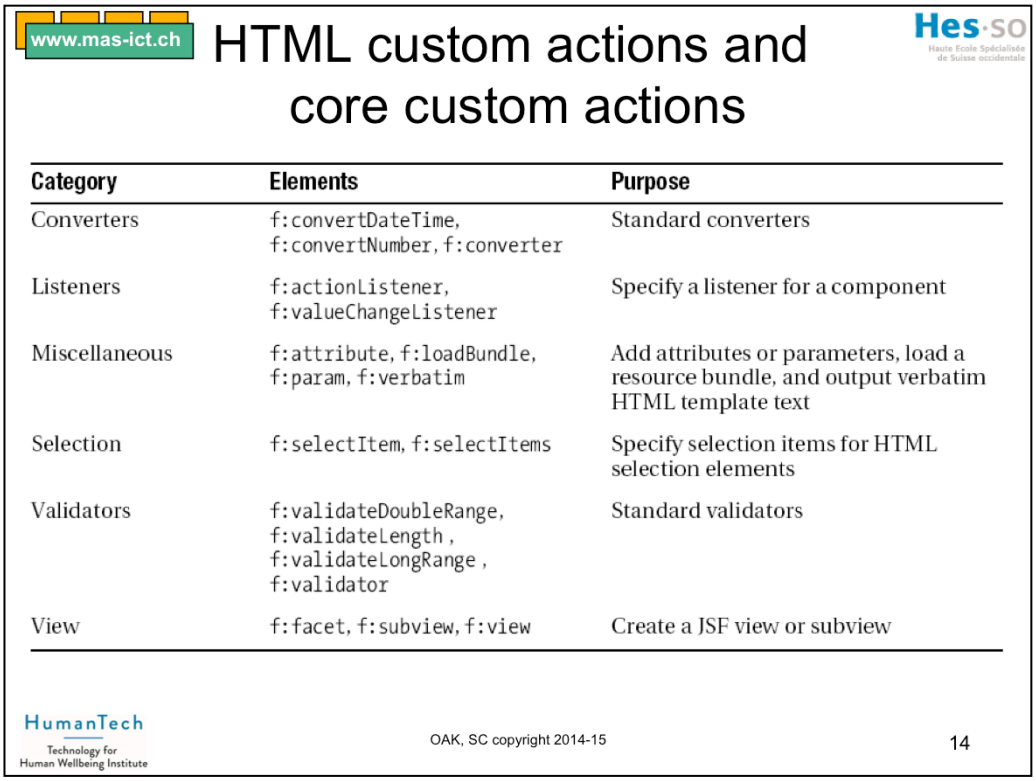
<%@ taglib uri="http://java.sun.com/jsf/core/" prefix="f" %>

Components

<h:commandButton id="submit" action="next" value="Submit" />

<h:inputText id="userName" value="#{GetNameBean.userName}" required="true" >

<h:outputText value="#{Message.greeting_text}" />



Category	Elements	Purpose
Converters	f:convertDateTime, f:convertNumber, f:converter	Standard converters
Listeners	f:actionListener, f:valueChangeListener	Specify a listener for a component
Miscellaneous	f:attribute, f:loadBundle, f:param, f:verbatim	Add attributes or parameters, load a resource bundle, and output verbatim HTML template text
Selection	f:selectItem, f:selectItems	Specify selection items for HTML selection elements
Validators	f:validateDoubleRange, f:validateLength , f:validateLongRange , f:validator	Standard validators
View	f:facet, f:subview, f:view	Create a JSF view or subview

The core custom actions create UI elements that are independent of the render kit. These actions are usually used in conjunction with the HTML actions.

Example

User input validation:

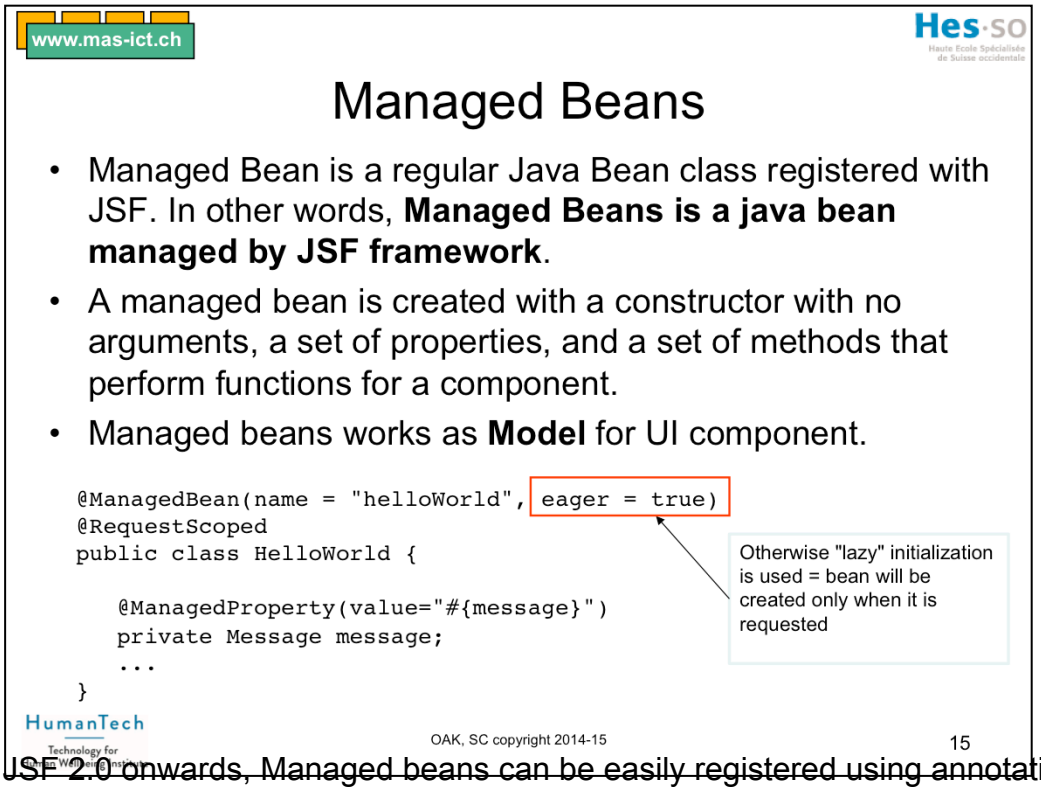
•If validation or type conversion is unsuccessful, a component specific FacesMessage instance is added to FacesContext. The message contains summary, detail and severity information

•Validation can also be delegated to a managed bean by adding a method binding in the validator attribute of an input tag.

•This mechanism is particularly useful for accomplishing form validation, where combinations of inputted values need to be evaluated to determine whether validation should succeed.

Standard/Built-in validation components

```
<h:inputText id="age" value="#{UserRegistration.user.age}">
        <f:validateLongRange maximum="150" minimum="0"/>
</h:inputText>
```



From JSF 2.0 onwards, Managed beans can be easily registered using annotations

@ManagedBean marks a bean to be a managed bean with the name specified in **name** attribute. If the name attribute is not specified, then the managed bean name will default to class name portion of the fully qualified class name. In our case it would be helloWorld.

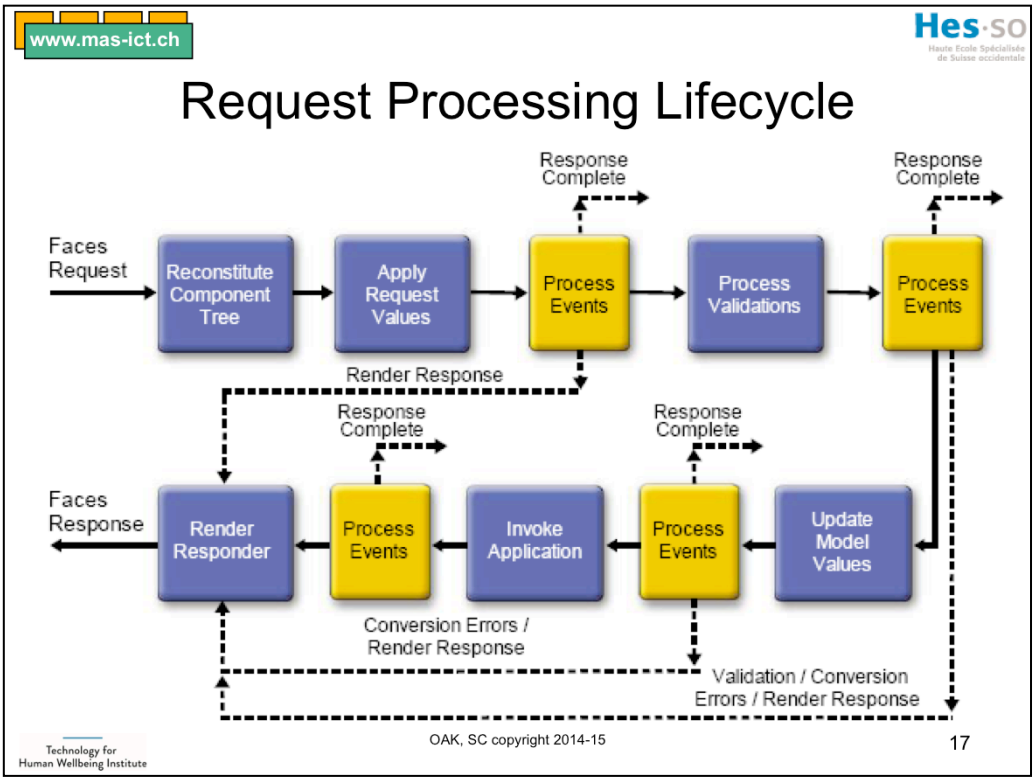
Another important attribute is **eager**. If eager="true" then managed bean is created before it is requested for the first time otherwise "lazy" initialization is used in which bean will be created only when it is requested.

Scope annotations set the scope into which the managed bean will be placed. If scope is not specified then bean will default to request scope.

Managed Bean Scopes

- Application
  - Application scope persists across all users' interactions with a web application
- Session
  - Session scope persists across multiple HTTP requests in a web application
- Flow
  - Flow scope persists during a user's interaction with a specific flow of a web application
- Request
  - Request scope persists during a single HTTP request in a web application
- Dependent
  - Dependent scope indicates that the bean depends on some other bean

- Application (javax.enterprise.context.ApplicationScoped): Application scope persists across all users' interactions with a web application.
- Session (javax.enterprise.context.SessionScoped): Session scope persists across multiple HTTP requests in a web application.
- Flow (javax.faces.flows.FlowScoped): Flow scope persists during a user's interaction with a specific flow of a web application. An application can have any number of flows. Each flow includes a set of pages and, usually, one or more managed beans scoped to that flow. Each flow has a starting point, called a start node, and an exit point, called a return node.
- Request (javax.enterprise.context.RequestScoped): Request scope persists during a single HTTP request in a web application.
- Dependent (javax.enterprise.context.Dependent): Indicates that the bean depends on some other bean.



The JSF Life Cycle (Figure shows the life cycle of a JSF page as it is processed by the JSF servlet.**)**

The up-right arrows you see in the blocks numbered 3, 5, 7, and 8 indicate that the cycle can be interrupted. That happens when the response to be generated doesn't contain any JSF component. In that case, JSF abandons the processing of the request by executing the FacesContext.responseComplete method. Let's go through the life-cycle phases one by one:

1. Restore View: The JSF servlet builds the view of the requested page as a component tree that contains the information associated with all components. It also saves the view in a FacesContext instance, thereby making it possible to repopulate the page if necessary—for example, when the user doesn't fill out a form as required. If the same page was displayed before and component states were saved, that information would also be taken into account. In this phase, JSF wires event handlers and validators (if any) to the components.

2. Apply Request Values: The JSF servlet goes through the component tree and executes each component's decode method, which extracts values from the request parameters and stores them locally in the component. It also automatically converts the parameters that are associated with object properties of non-string types. Conversion errors cause error messages to be queued to the FacesContext object. In some cases, typically when the user clicks on controls, the servlet also generates request events and queues them to FacesContext.

3. Process Request Events: The servlet calls the processEvent method for each component with one or more queued events. Each component decides to handle the events or delegate their handling to event handlers. In any case, the servlet proceeds with the next phase if all executed processEvent methods return false. Otherwise, it jumps directly to the Render Response phase.

4. Process Validation: The servlet invokes the validate methods of the validators that had been registered during the Restore View phase. For each validate method that returns false, the servlet queues an error message to the FacesContext.

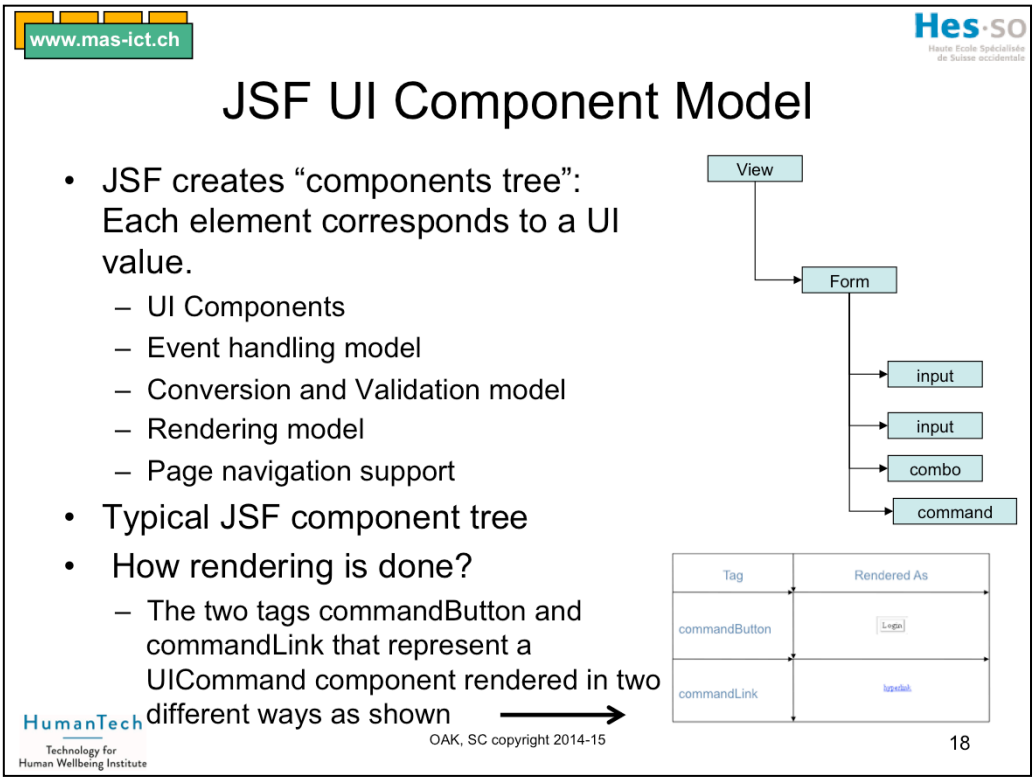
5. Process Events: If validation or conversion errors are generated during the Process Validation phase, control jumps directly to the Render Response phase.

6. Update Model Values: Each UI component can be linked to a field in a Java object called the *model object. During this phase, the values of the linked components are* copied to the corresponding fields of the model object by executing the component method updateModel, which also does type conversions when necessary. Conversion errors cause error messages to be queued to FacesContext.

7. Process Events: If it turns out that conversion errors were generated during phase 6, control jumps directly to the Render Response phase.

8. Invoke Application: During this phase, the servlet processes the application-level events by executing the corresponding handlers. When the user submits a form or clicks on a link of a JSF application, the JSF servlet generates a corresponding application-level event. One of the tasks you have to do when developing a JSF application is to assign a handler (e.g., a JSP page) to each one of the possible application events.

9. Render Response: The servlet creates a response component tree and delegates the rendering of the page to server. Each component renders itself as server goes through the corresponding JSF tags. At the end of this phase, the state of the response is saved so that the servlet can access it during the Restore View phase of subsequent requests to the same page.



JSF creates a “components tree”: Each element corresponds to a UI value.

UI Components:

UIComponent/UIComponentBase: Base class for all user interface components. Standard UIComponent Subclasses: UICommand, UIForm, UIOutput, UIGraphic, UIInput, UIPanel, UIParameter, UISelectBoolean, UISelectMany, UISelectOne.

•UIForm: Encapsulates a group of controls that submit data to the application. This component is analogous to the form tag in HTML.

•UIInput: Takes data input from a user. This class is a subclass of UIOutput.

•UICommand: Represents a control that fires actions when activated.

•UIOutput: Displays data output on a page.

•UIMessage: Displays a localized message.

Event handling model

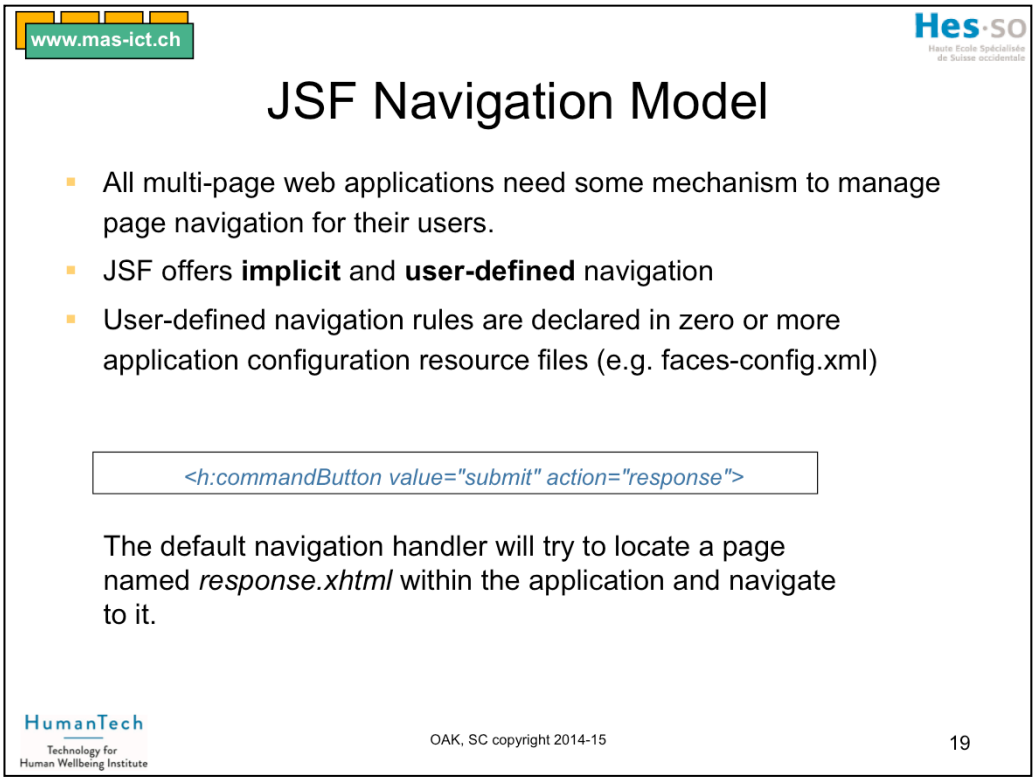
Follows JavaBeans™ Specification design and naming patterns, standard events and listeners:

•ActionEvent—UICommand component activated by the user

•ValueChangedEvent—UIInput component whose value was just changed

Render Classes

For every UI Component that a render kit supports, the render kit defines a set of renderer classes. Each renderer class defines a different way to render the particular component to the output defined by the render kit. For example a UISelectOne component has three different renderers. One of them renders the component as a set of radio buttons. Another renders the component as a combo box. The third one renders the component as a list box. Each JSP custom tag defined in the standard HTML render kit is composed of the component functionality(defined in UIComponent class) and the rendering attributes(defined by the renderer class)



How Navigation is done

•When a button or hyperlink is clicked the component associated with it generates an action event.

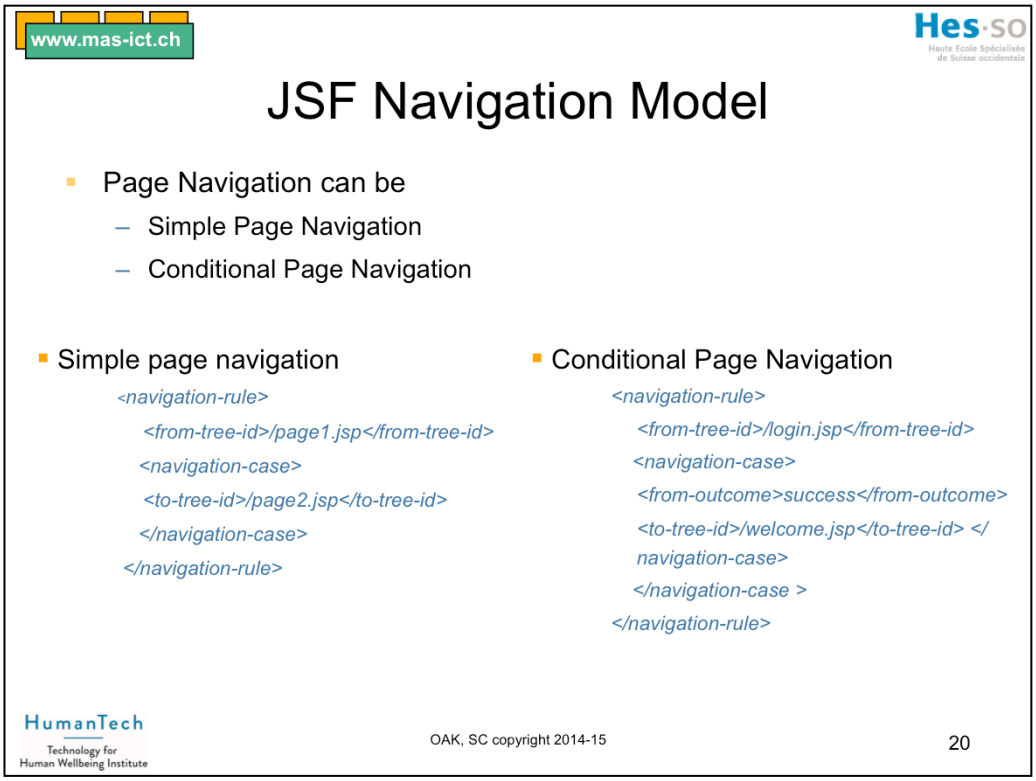
•This event is handled by the default ActionListener instance, which calls the action method referenced by the component that triggered the event.

•This action method is located in backing bean and is provided by application developer.

•This action method returns a logical outcome String which describes the result of the processing.

•The listener passes the outcome and a reference to the action method that produced the outcome to the default NavigationHandler.

•The NavigationHandler selects the next page to be displayed by matching the outcome or the action method reference against the navigation rules in the application configuration resource file.



How Navigation is done

•When a button or hyperlink is clicked the component associated with it generates an action event.

•This event is handled by the default ActionListener instance, which calls the action method referenced by the component that triggered the event.

•This action method is located in backing bean and is provided by application developer.

•This action method returns a logical outcome String which describes the result of the processing.

•The listener passes the outcome and a reference to the action method that produced the outcome to the default NavigationHandler.

•The NavigationHandler selects the next page to be displayed by matching the outcome or the action method reference against the navigation rules in the application configuration resource file.

Benefits

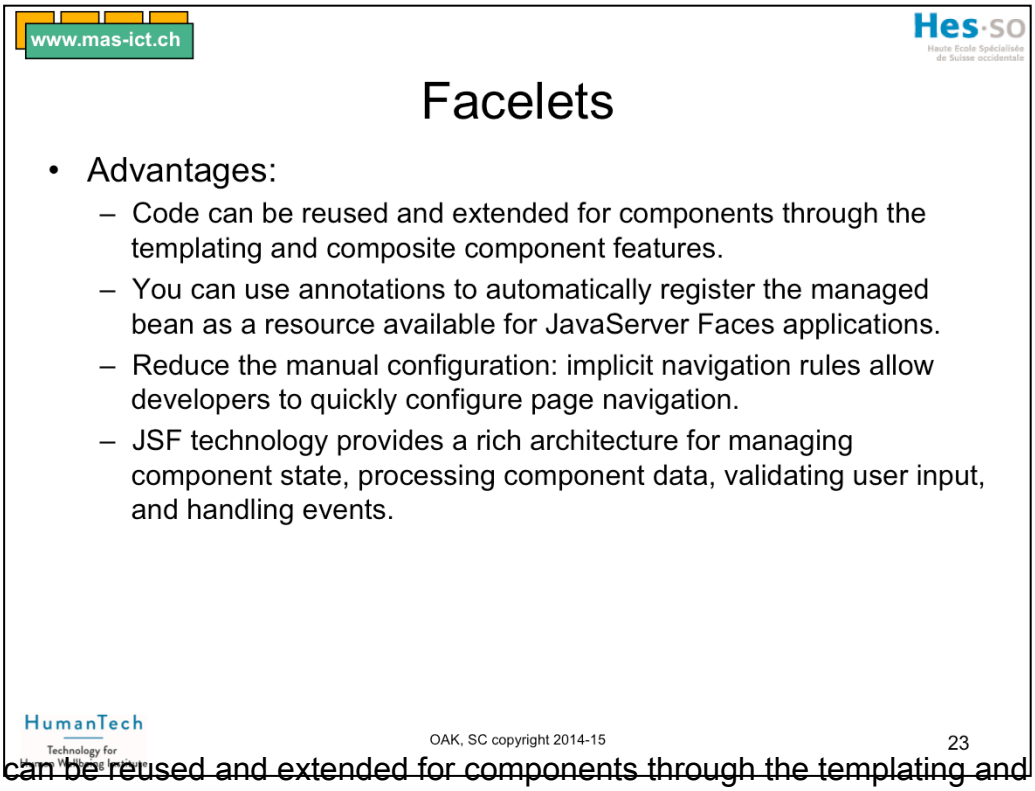
- Separation of logic from presentation
- Page authors with no programming expertise can use JavaServer Faces technology tags in a web page to link to server-side objects without writing any scripts.

JEE 5 JEE 7

- **Facelets technology** is the preferred presentation technology for building JSF technology–based web applications.

Facelets

- The term Facelets refers to the view declaration language for JavaServer Faces technology
- This language is used to build JavaServer Faces views using HTML style templates and to build component trees.
- Features:
  - Use of XHTML for creating web pages
  - Support for Facelets tag libraries in addition to JavaServer Faces and JSTL tag libraries
  - Support for the Expression Language (EL)
  - Templating for components and pages



Code can be reused and extended for components through the templating and composite component features.

You can use annotations to automatically register the managed bean as a resource available for JavaServer Faces applications. In addition, implicit navigation rules allow developers to quickly configure page navigation (see Navigation Model for details). These features reduce the manual configuration process for applications.

Most important, JavaServer Faces technology provides a rich architecture for managing component state, processing component data, validating user input, and handling events.

JSF and Expression Language

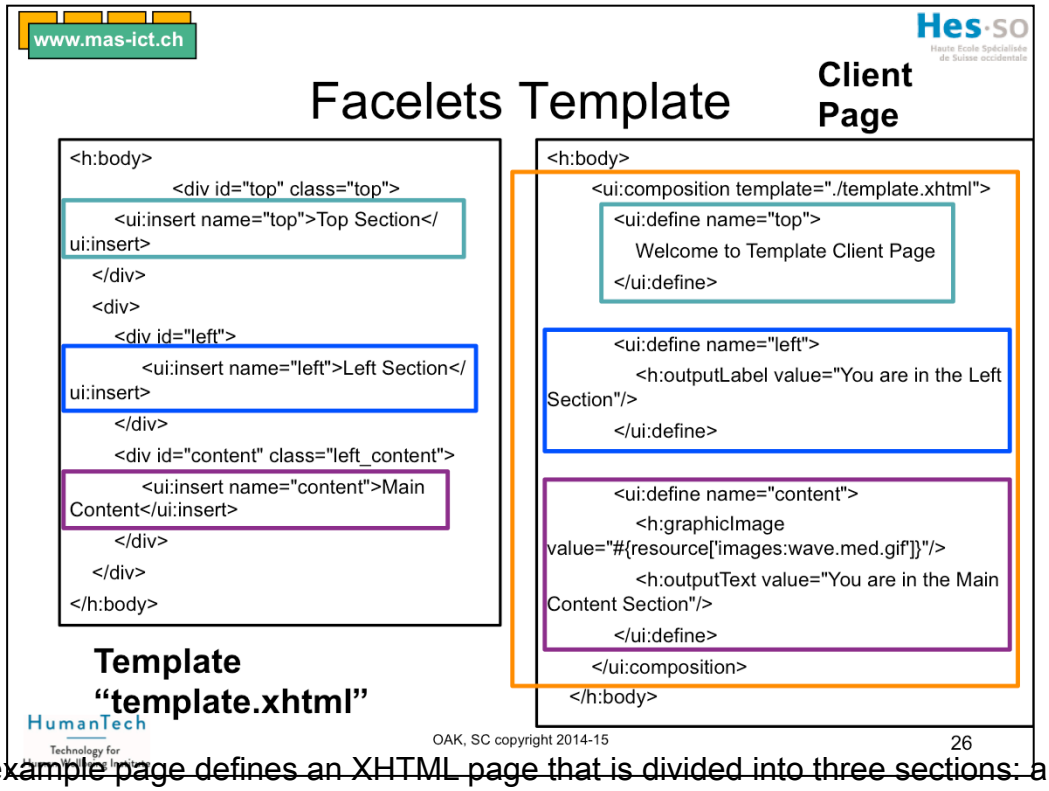
- Expression Language (EL), provides an important mechanism for enabling the presentation layer (facelets) to communicate with the application logic.
- To bind component values and objects to managed bean properties or to reference managed bean methods from component tags.
- JSF EL expressions can refer to the following objects and their properties or attributes:
  - JavaBeans components
  - Collections
  - Java SE enumerated types
  - Implicit object

You can find a list of implicit objects in jsf here: http://incepttechnologies.blogspot.ch/p/jsf-implicit-objects.html

Facelets Template

- Templating is a useful Facelets feature that allows you to create a page that will act as the base, or **template**, for the other pages in an application.
- Specifics Facelets tags are used for templating. Ex:
  - **<ui:insert>** Insert content into a template.
  - **<ui:composition>** Defines a page composition that optionally uses a template.
  - **<ui:define>** Defines content that is inserted into a page by a template.
  - …

More information on using facelets templates here: http://docs.oracle.com/javaee/7/tutorial/doc/jsf-facelets004.htm



The example page defines an XHTML page that is divided into three sections: a top section, a left section, and a main section. The same structure can be reused for the other pages of the application.

The client page invokes the template by using the ui:composition tag.

JSF as MVC

- **Model**: The Services/DAOs plus the entities they produce and consume. The entry point to this is the managed bean, but in Java EE (of which JSF is a part) these artifacts are typically implemented by **EJB** and **JPA** respectively.
- **View**: The UI components and their composition into a full page. This is fully in the domain of JSF and implemented by **JSF UIComponents** and **Facelets** respectively.
- **Controller**: In JSF one doesn't write this controller, but it's already provided by the framework. It is the **FacesServlet**).

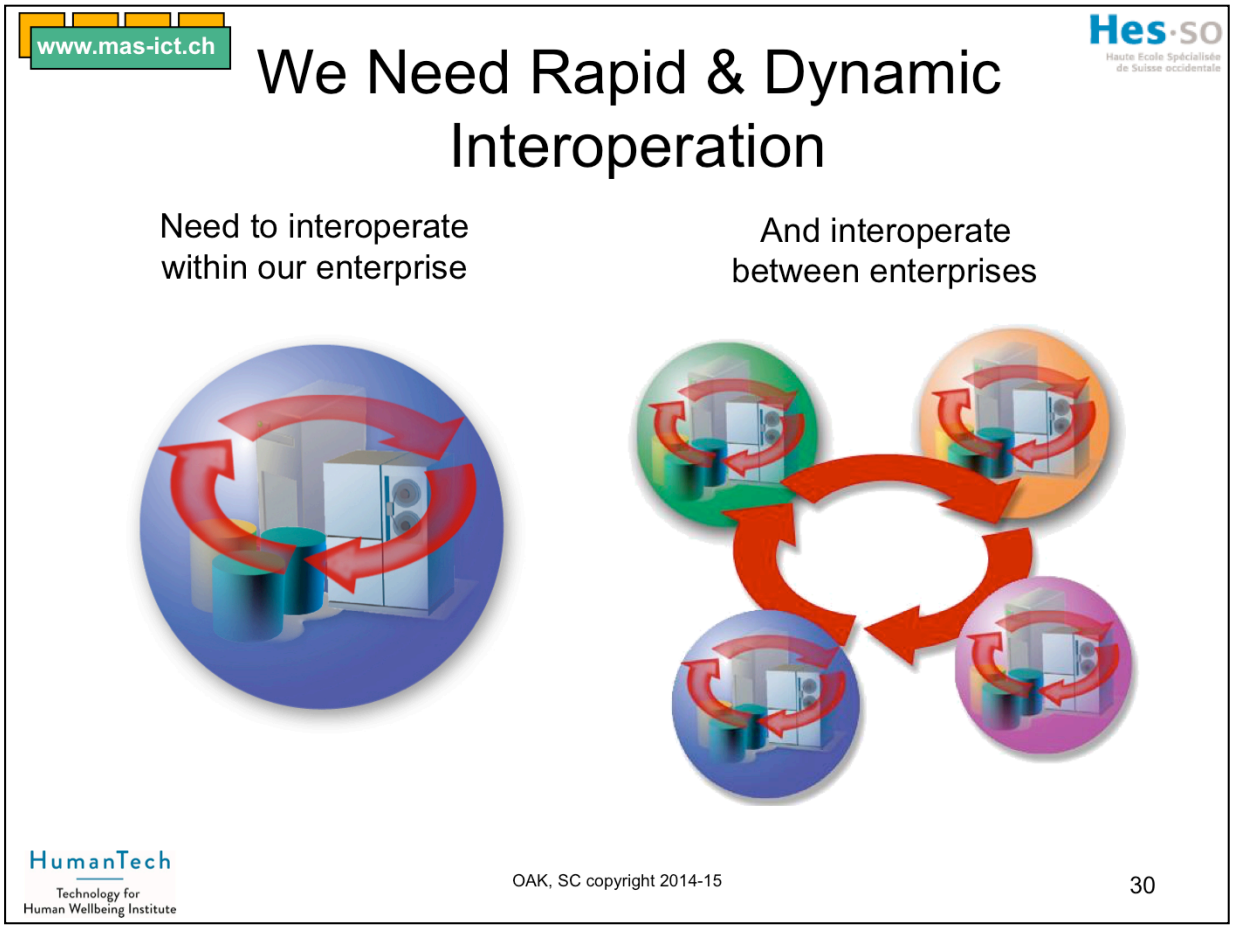
Applications built with JSF are intended to follow the model-view-controller (MVC) architectural pattern. The JSF framework implements the Model-View-Controller (MVC) architecture ensuring that applications are well designed and easier to maintain.

Service Oriented Architecture

SOA & WEB SERVICES

Barriers to eBusiness

- Isolated business information & systems
  - Diverse & incompatible technologies
  - Business functions locked in multiple, disconnected systems
  - Within and across business boundaries
- Drastically changing business relationships
  - Frequent reorganizations & mergers
  - Emerging alternative channels
  - Constant new business partners
- Inflexible business processes
  - Incompatible mix of systems, users and devices
  - Systems that can't keep up with business change



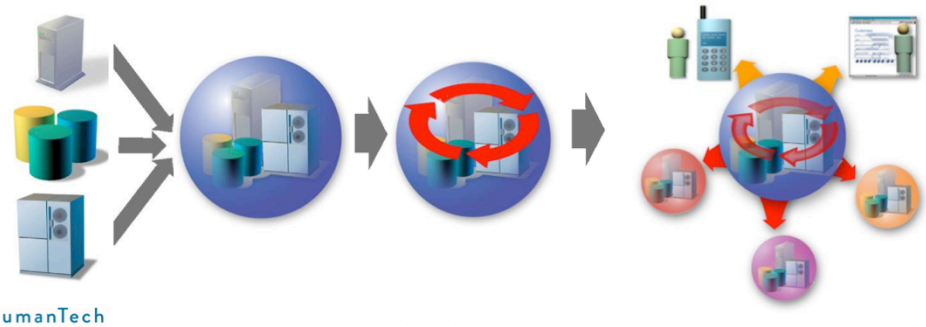
In the past, the primary drivers for the requirement for application integration arose inside the enterprise, and were associated with things like the implementation of ERP (**Enterprise Resource Planning**) packaged applications. Solutions to meet these requirements have been generally referred to as being for **enterprise application integration** (EAI). While these drivers continue to operate, a new class of drivers has begun to come to the fore, which arises from the integration that is taking place across multiple enterprises, with the Internet as an integration medium, primarily in support of business-to-business e-commerce.

This kind of externally focused integration is heavily associated with e-business, both business to consumer and business to business. But the impact is more fundamental than just new ways of business communication. Enterprises are increasingly finding that innovative new business models are springing from the opportunity to make major changes to business processes, major shifts in the roles each enterprise takes in the overall business process, and in the ways those roles are effected by new information technology inside and across these enterprises.

Although EAI solutions will continue to be required, and to exist as a distinct market, a new category of solutions that it partly overlaps is now beginning to emerge, which Giga calls "**Internet Application Integration**."

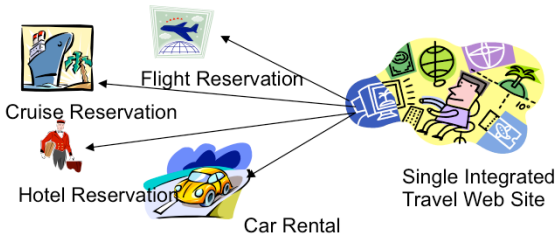
What's Needed to Solve the Problem?

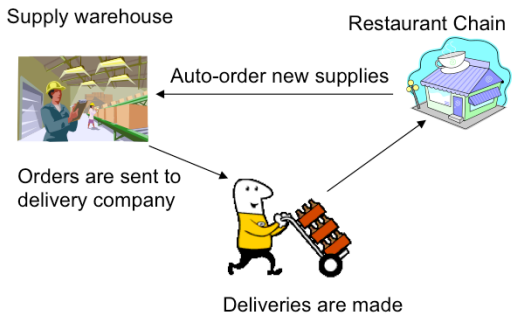
- Make business functions readily shareable
- Link them together into a flexible process flow
- Deliver them in a relevant way, in the right format
- Make them discoverable and available to others anywhere



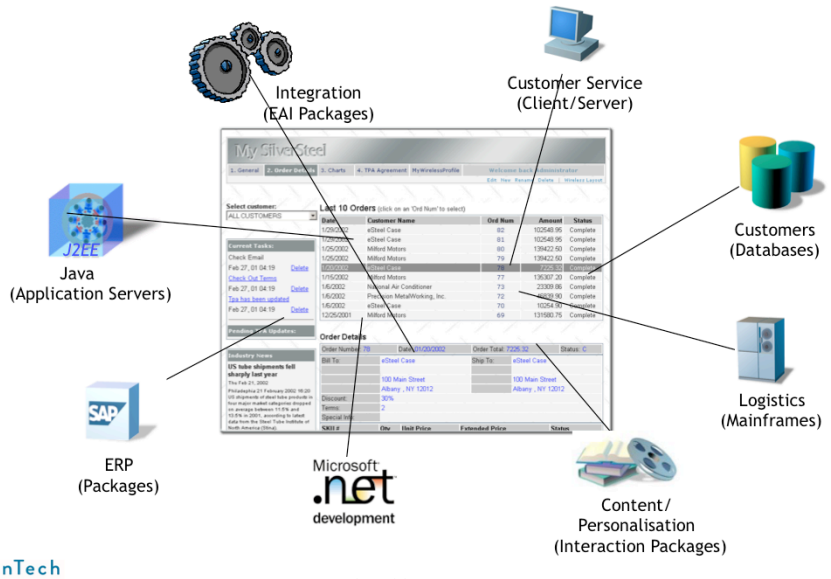
Business process

- Extension from intra to inter enterprise
- Principe of self service aggregation at partners level
- Focalization on "core business“
- Outsourcing for the rest of activities

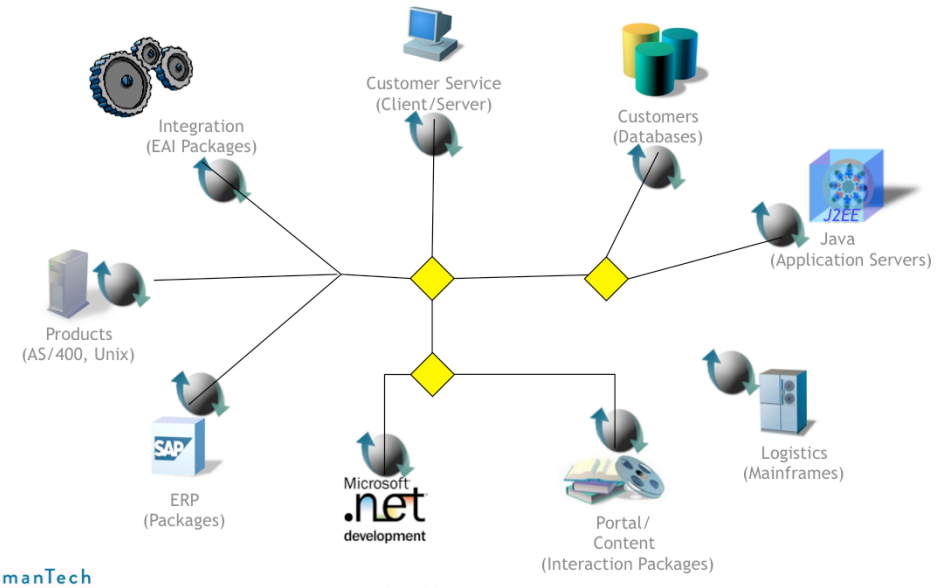




Business process



Business process



Possible solutions:

- Until now:
  - Distributed Architecture
    - Component-based architecture
- From now:
  - Service-oriented Architecture (SOA)

Client-server system architecture

vs.

Client-server distributed computing

Source: http://n-tier.com/articles/csoverview.html

Client Server - As the name says you have a client and a server, client will ask for some service, which the server will deliver.
Distributed Architecture - This is when we have more than 1 machine involved in system. Say for a ecommerce project you may have a app server which will server UI, but you could have some other server to process orders. You could have more than 1 database. Client server is distributed to an extent. But usually it is tied to 2 or 3 tier. Distributed could be n-tier.

In the client-server system architecture, the terms clients and servers refer to computers, while in the client-server distributed computing paradigm, the terms refer to processes.

Client-server architectures: Distributed services which are called on by clients. Servers that provide services are treated differently from clients that use services

Distributed object architectures: No distinction between clients and servers. Any object on the system may provide and use services from other objects

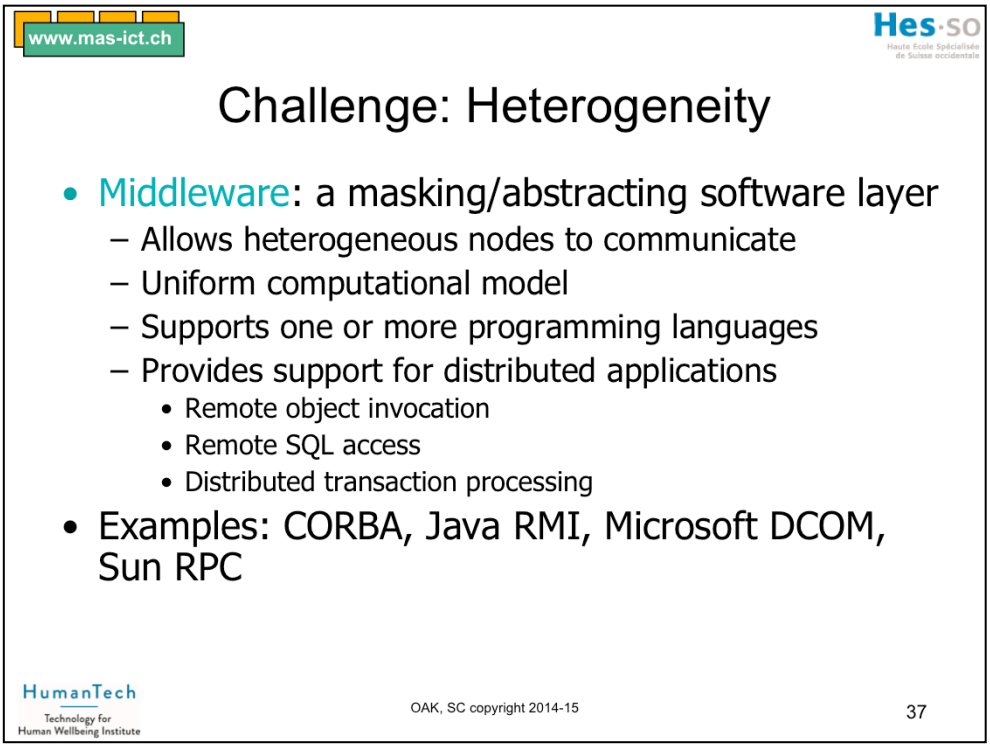
Distributed System Definitions

- "A distributed system is a collection of independent computers that appear to the users of the system as a single computer." [Tanenbaum]
- "A distributed system is a collection of autonomous computers linked by a network with software designed to produce an integrated computing facility."
  [Coulouris, Dollimore, Kindberg]
- "A system of multiple autonomous processing elements, cooperating in a common purpose or to achieve a common goal." [Burns & Wellings 1997]
- Autonomy requirement excludes thin clients
- Purpose requirement excludes the Internet and general-purpose networks

"A system in which components located at networked computers communicate and coordinate their actions only by message passing."
[Coulouris et al. 2001]

Excludes tightly coupled systems, i.e., systems with shared memory or a shared clock

"A system that consists of a collection of two or more independent computers which coordinate their processing through the exchange of synchronous or asynchronous message passing."



Component-based architecture:

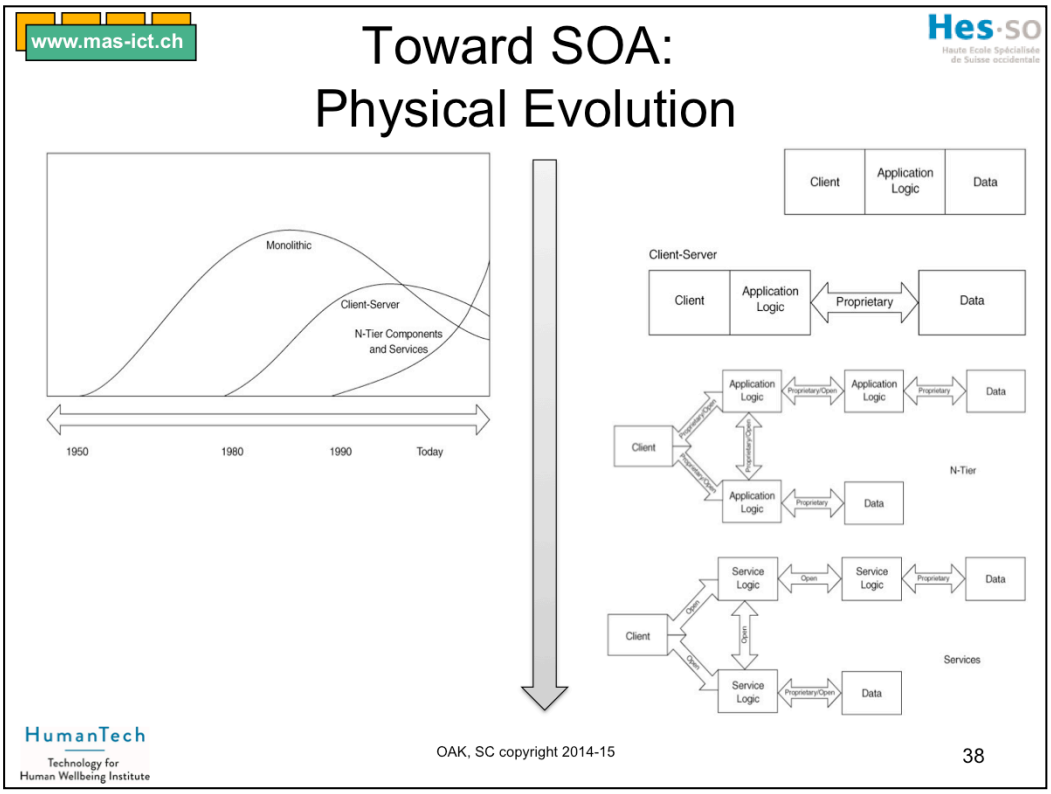
A component is a software object, meant to interact with other components, encapsulating certain functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behavior common to all components within an architecture.

1 The goal of generative and component-based software engineering is to increase productivity, quality, and time-to-market in software development thanks to the deployment of both standard componentry and production automation. One important paradigm shift implied here is to build software systems from standard componentry rather than "reinventing the wheel" each time. This requires thinking in terms of system families rather than single systems. Another important paradigm shift is to replace manual search, adaptation, and assembly of components with the automatic generation of needed components on demand. Generative and component-based software engineering seeks to integrate domain engineering approaches, component-based approaches, and generative approaches.

2 A component model is probably used for the developing and executing of components. This model defines a framework, which defines structural requirements for connection- and composition options as well as behaviour-oriented requirements for collaboration options to the components. Beyond that a component model provides an infrastructure which implements frequently used mechanism like persistence, message-exchange, security and versioning. The idea is to build exchangeable software units through clearly defined interfaces. Different manufactures offer platforms like DCOM, JavaBeans, Enterprise JavaBeans, and CORBA.

CORBA is an international standard for an Object Request Broker - middleware to manage communications between distributed objects. Several implementation of CORBA are available.

DCOM is an alternative approach by Microsoft to object request brokers.



Physical Evolution: The Evolution in Computer System Deployment

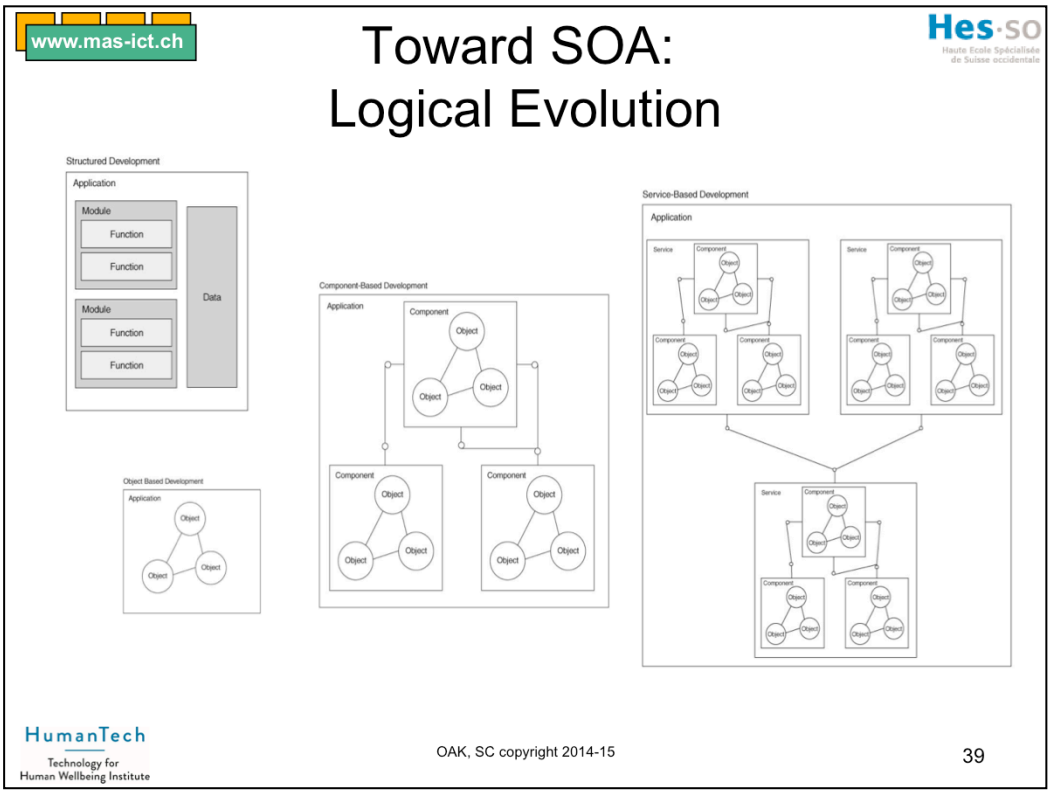
Monolithic ➔ Client-Server ➔ *N*-Tier Development ➔ World Wide Web

Monolithic: From the 1950s through the 1980s, programmers developed systems mostly on closed, monolithic mainframe systems, using languages such as COBOL and FORTRAN. These procedural methods for systems development, represented in Figure, required that all parts of a software system run on the same machine and that the program have direct access to the data upon which it was acting. The software was developed in a handful of languages and required an intricate knowledge of the physical storage of the data.

Client-Server: In the 1980s, client-server computing fostered a leap forward in the decomposition of system components. As Figure shows, client-server developers separated the processor of the data (client) from the keeper of the data (server). This separation was both logical and physical. The client process retrieved the data from a server across a network connection. This enhanced flexibility, because the data was encapsulated into a database server. Database designers and administrators wrapped the data in referential integrity constraints and stored procedures to ensure that no client program could corrupt the data. The relational database also insulated the client from having to know the physical layout of the data.

***N*-Tier Development :** *N*-tier development, which started in the early 1990s, reduced the coupling between the client and the business logic. As Figure shows, developers created a business-logic layer that was responsible for accessing data and providing application logic for the client. This additional layer also allowed multiple clients to access the business logic. The clients converse with the business logic tier in less interoperable protocols, such as DCOM.

World Wide Web: With the introduction of the World Wide Web, *n*-tier development has been taken a step further. As Figure shows, developers create independent services accessible through the firewall. Services are modules that support one discrete function. For instance, a checking account service supports all the functions of managing checking accounts, and no other function. Clients interact with services using open technologies. Services are built from standard technologies, such as the ones used by the Internet. Applications are assemblies of services. Services do not "know" into which applications they are assembled.



Logical Evolution:

Structured Design ➔ Object-Oriented Development ➔ Component-Based Development ➔ Service-Based Development

Structured Design: *Structured design and development*, diagrammed in Figure, involves decomposing larger processes into smaller ones. Designers break down larger processes into smaller ones to reduce complexity and increase reusability. Structured design addresses the behavior portion of a software system separately from the data portion. Breaking down a program's structure helps develop more complex applications, but managing data within the application is difficult, because different functions act on much of the same data.

Object-Oriented Development: *Object-oriented development*, represented by Figure, allows software designers and developers to encapsulate both data and behavior into classes and objects. This places the data near the code that acts on the data and reduces the dependencies between objects. An additional benefit to object orientation is that software structures more easily map to real-world entities. Object-based development advances the information-hiding principles of software design by providing more support for hiding behavior and data.

Component-Based Development: Components are larger and coarser grained than objects and do not allow programs to access private data, structure, or state. *Component-based development* allows developers to create more complex, high-quality systems faster than ever before, because it is a better means of managing complexities and dependencies within an application.

Service-Based Development: With *service-based development,* represented in Figure, services are usually components wrapped in a service layer. A service tends to aggregate multiple components into a single interface and is thus coarser-grained. Also, the consumer of a service does not know the location or the interface of the service until runtime. This is called "late binding." The consumer finds the location of the service at runtime by looking it up in a registry, which also contains a pointer to the service contract. The contract describes the interface for the service and how to invoke it.

Toward SOA: SOA shifts the way we think

Traditional Distributed Computing	Service Oriented Architecture
Designed to last	Designed to change
Tightly Coupled	Loosely Coupled, Agile and Adaptive
Integrate Silos	Compose Services
Code Oriented	Metadata Oriented
Long development cycle	Interactive and iterative development
Cost centered	Business centered
Middleware makes it work	Architecture makes it work
Favors Homogeneous Technology	Favors Heterogeneous Technology

INTRODUCTION TO SOA

What is a Service

- A facility supplying some public demand
- The work performed by one that serves
- HELP, USE, BENEFIT
- A Windows Service?
  - EventLog, DHCP Client
- Software Service?
  - Distribution Service, Alert Service
  - Security Service, Log Service
- Business Service?
  - Common Operational Picture, Navigation
  - Accounts Receivable, Customers

http://www.merriam-webster.com/dictionary/serves
http://www.merriam-webster.com/dictionary/help
http://www.merriam-webster.com/dictionary/use
http://www.merriam-webster.com/dictionary/benefit

In economics and marketing, a service is the non-material equivalent of a good. Service provision has been defined as an economic activity that does not result in ownership, and this is what differentiates it from providing physical goods. It is claimed to be a process that creates benefits by facilitating either a change in customers, a change in their physical possessions, or a change in their intangible assets.

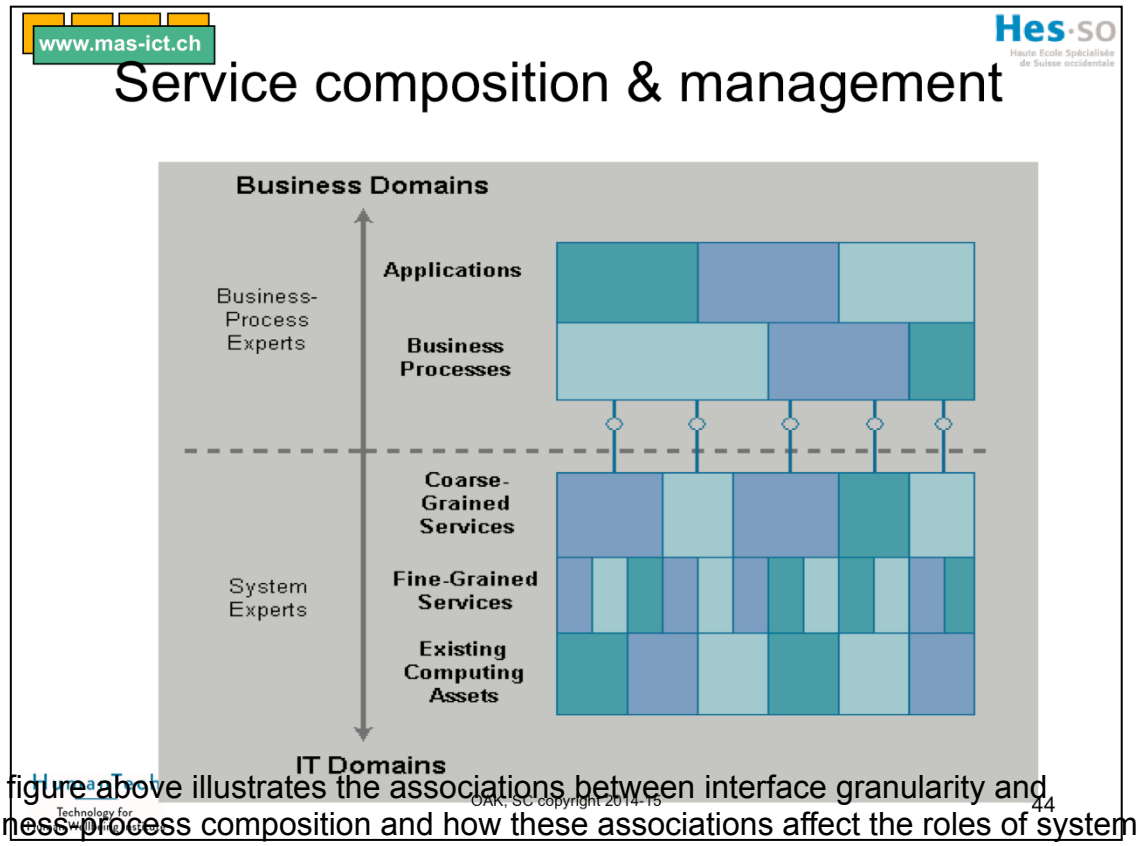
Service Granularity

- Service granularity refers to the scope of functionality a service exposes.
  - Fine-grained services provide a small amount of business-process usefulness, such as basic data access.
  - Coarse-grained services are constructed from fine-grained services that are intelligently structured to meet specific business needs.

Services and Systems

A service is a program you interact with via message exchanges. Services are built to last. Encompass a business perspective. Stability and robustness are critical

A system is a set of deployed services cooperating in a given task. Systems are built to change. Adapt to new services after deployment

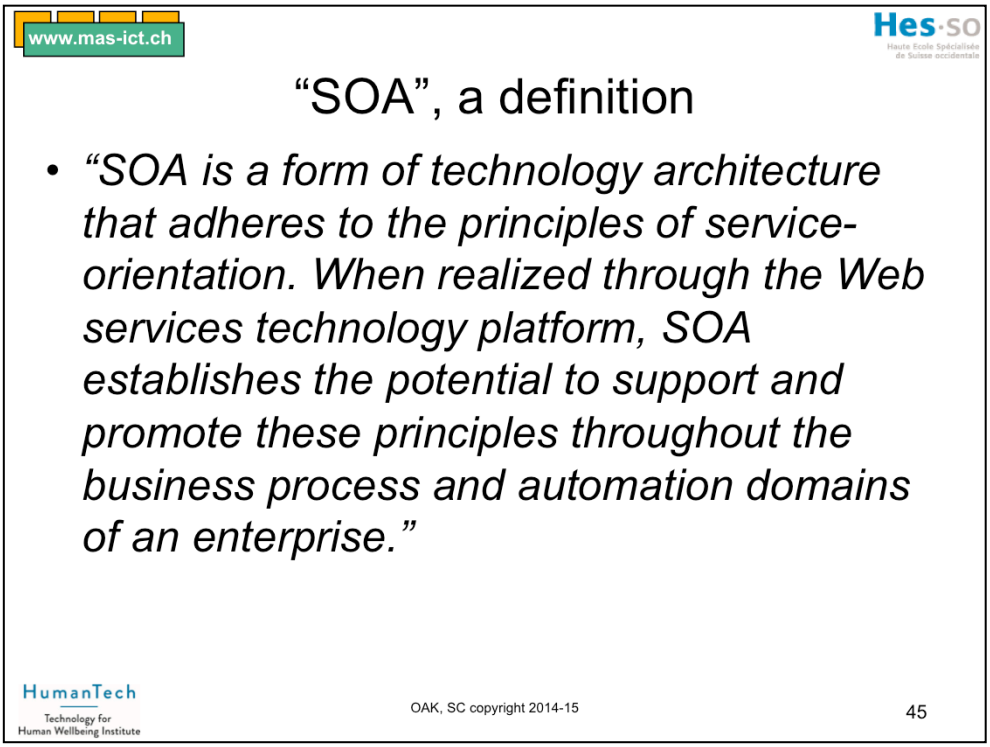


The figure above illustrates the associations between interface granularity and business process composition and how these associations affect the roles of system experts and business experts.

Service Management

There are three levels of service management:

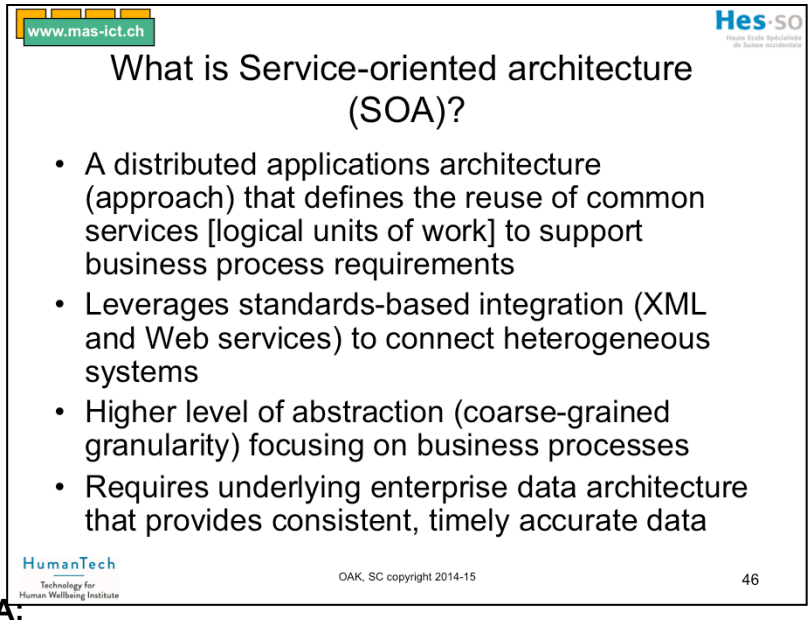
1. Management of the services and their interfaces
2. Management of the underlying system
3. Management of the interaction between the services and the underlying system



SOA – Definition

1. An approach for building distributed computing systems based on encapsulating business functions as services that can be easily accessed in a loosely coupled fashion.
2. Service-oriented architecture (SOA) is an approach to loosely coupled, protocol independent, standards-based distributed computing where software resources available on the network are considered as Services.
3. SOA is believed to become the future enterprise technology solution that promises the agility and flexibility the business users have been looking for by leveraging the integration process through composition of the services spanning multiple enterprises.
4. Service-Oriented Architecture is an IT strategy that organizes the discrete functions contained in enterprise applications into interoperable, standards-based services that can be combined and reused quickly to meet business needs.
5. The goal of a Service-Oriented Architecture (SOA) is to enable organizations to realize business and technology advantages through a combination of process innovation, effective governance, and a technology strategy that revolves around the definition and re-use of services.

While SOA is a long-term strategy that requires sustained focus on transforming the way IT is delivered, it must respond to immediate business initiatives. In consequence, the benefits of SOA will only be realized if you preserve a balance between the long-term goals and the shorter-term needs of the business. You can maintain that balance by instituting a set of organizational, financial, operational, design, and delivery practices from the outset of the SOA initiative.



The Vision for SOA:

Service orientation will encapsulate and componentized processes and systems: Help manage complexity, Permit controlled change, Support continuous improvement.

Business capabilities and business processes will be modelled as services:

1. Organizations will expose *touch points* into these processes to both internal and extra-organizational actors
2. Allows automation of processes that have defied automation until now

Service Oriented Architecture (SOA) based application development and integration solves many issues and short-comings of the traditional approaches mentioned earlier. SOA describes a set of well-established patterns that help a client application connect to a service. These patterns represent mechanisms used to describe a service, to advertise and discover a service, and to communicate with a service.

Most communication middleware systems, such as RPC, CORBA, DCOM, EJB, and RMI, rely on similar patterns. However, there some differences between each implementation, and weaknesses among them. **Primary issue has been around acceptable standards and interoperability**. SOA has built upon these technologies, and has tried to eliminate apparent weaknesses. **Each of these technologies has a fixed granularity, function for RPC, object for CORBA, etc. Services do not have this fixed granularity.** Instead a service can be a function, an object, an application, etc. SOA is therefore adaptable to any existing system, and does not force the system to conform to any particular level of granularity.

SOA helps information systems to move towards a “leave-and-layer” architecture. Instead of replacing the existing architectures, it helps to reuse all existing systems and applications in the company and transform them into agile services. SOA not only covers information from packaged apps, customer apps and legacy systems, but also the functionality/data from IT Infrastructure like security, content management, search, etc. A significant reduction of the time to develop new apps based on SOA was the ease at which they could add in functionality that came from these IT infrastructure services.

•SOA is an architectural approach that defines the reuse of common services to support growing business requirements

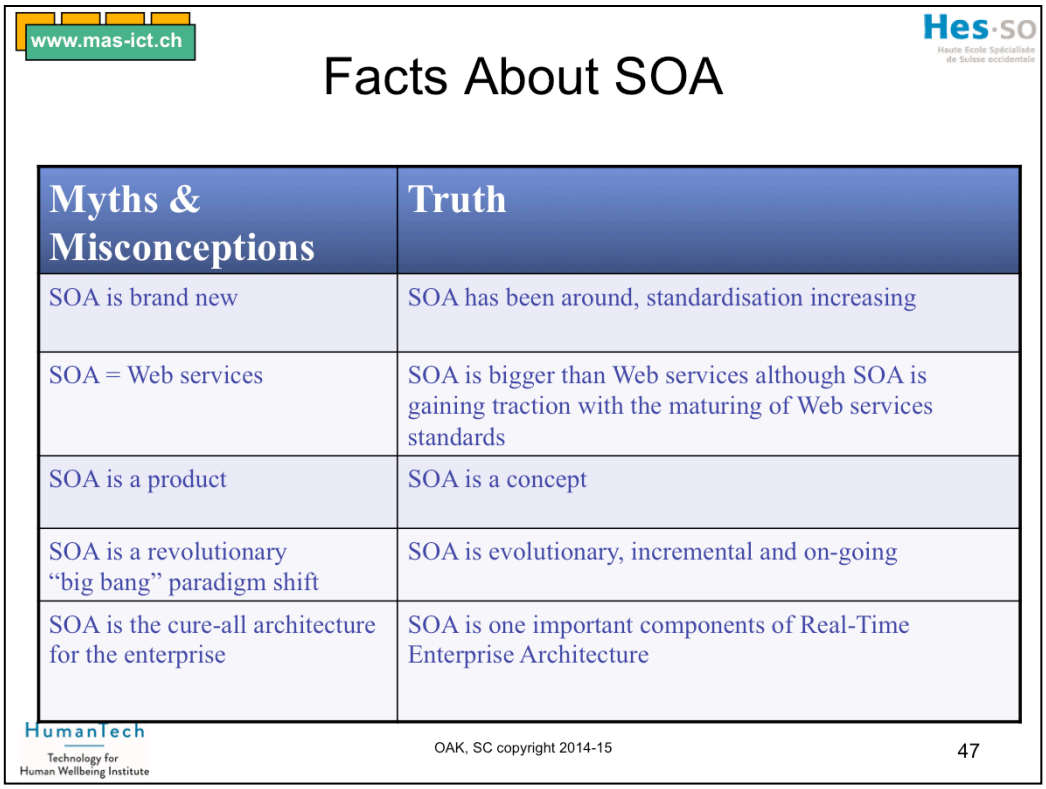
•Collection of services that communicate another via messages

•SOA is a higher level of application development (also referred to as coarse-grained granularity) mask the underlying technical complexity of the IT environment

•Services are loosely coupled, have well-defined, platform-independent interfaces, and are reusable

•Services provide access to data, business processes and IT infrastructure, ideally in an asynchronous manner

•SOA leverages standards-based integration (XML and Web services) to connect heterogeneous systems



What is SOA

•A generic definition of SOA could be: SOA is a development paradigm (and IT strategy) that breaks down monolithic applications into discrete business functions, leveraging standards to improve their interoperability, and can be reused to build composite applications to quickly satisfy new business requirements

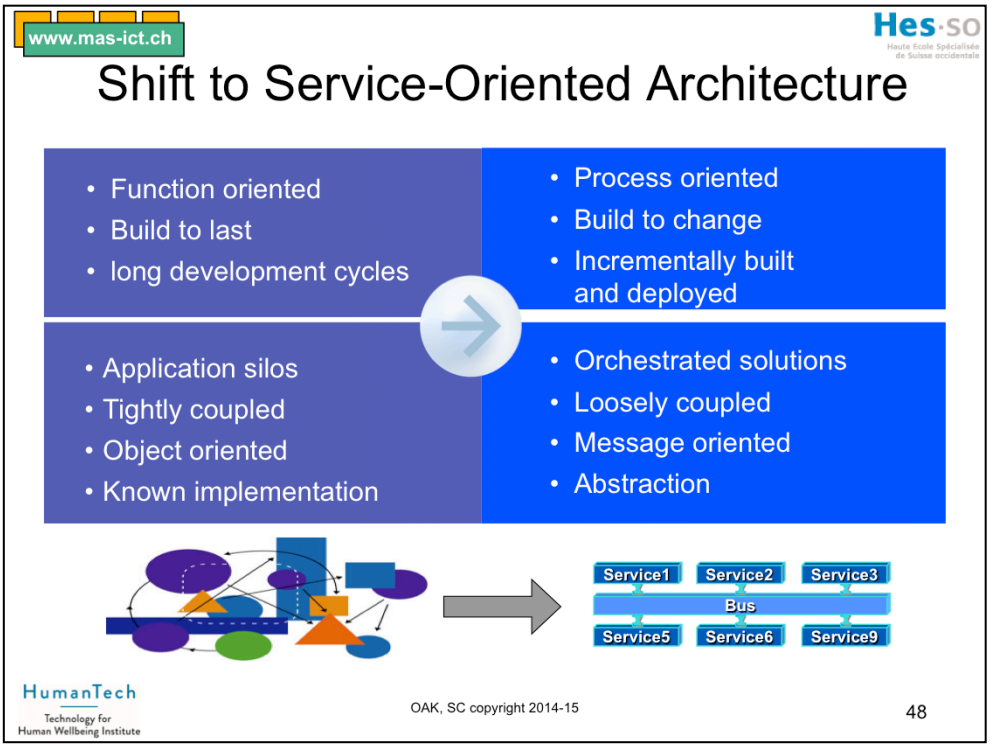
•Relevant points about SOA: SOA is not a product but an architectural paradigm. There is a lot of marketing hype around SOA

So What? Isn't this the same as OO, CORBA, DCOM, etc?

•It's the same goal: re-usability

•Key Difference is platform, vendor, operating system, language ***neutrality***

•***Neutrality*** allows re-use in heterogeneous environment



Before: Traditional development

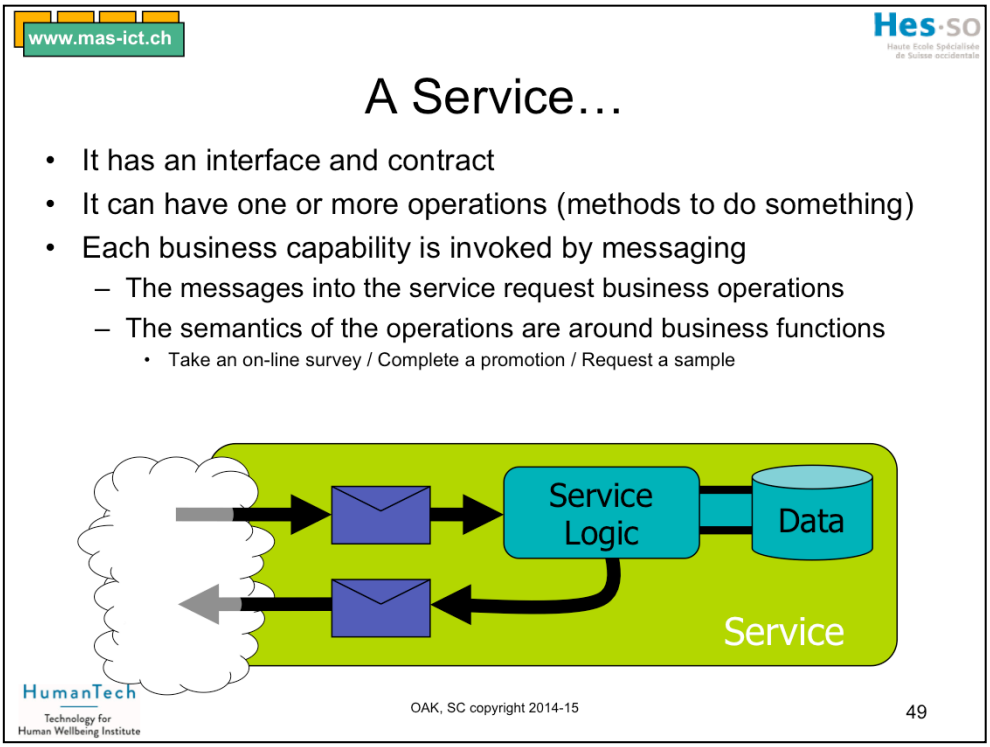
Monolithic application development, Point-to-point integration, Hard-wired

After: SOA / ESB

Distributed applications/modules, Backbone-based integration, reusable services

ESB : http://en.wikipedia.org/wiki/Enterprise_Service_Bus

In computing, an enterprise service bus (ESB) consists of a software architecture construct which provide fundamental services for complex architectures via an event-driven and standards-based messaging engine (the bus). Developers typically implement an ESB using technologies found in a category of middleware infrastructure products, usually based on recognized standards.

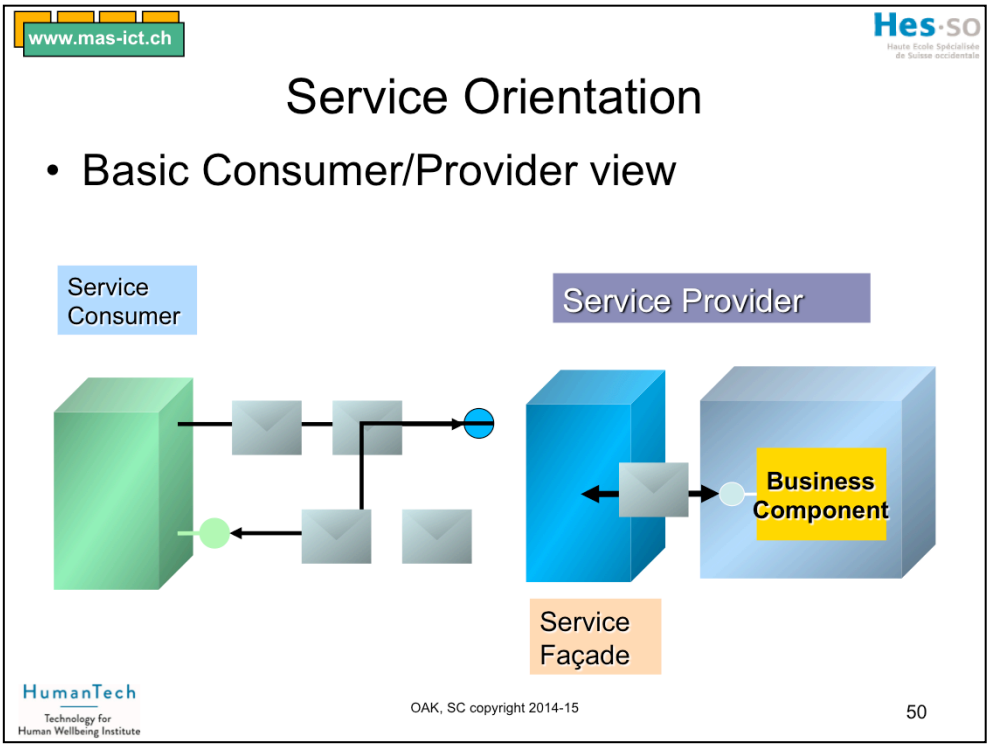


SOA Artifacts

The artifacts in a service-oriented architecture are:

•**Service:** A service that is made available for use through a published interface that allows it to be invoked by the service consumer.

•**Service description:** A service description specifies the way a service consumer will interact with the service provider. It specifies the format of the request and response from the service. This description may specify a set of preconditions, post conditions and/or quality of service (QoS) levels.



SOA Characteristics

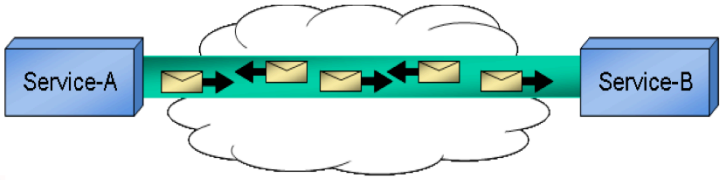
- Reusable logic is divided into services
- Services abstract underlying logic
- Services are composable
- Services are autonomous
- Services share a formal contract
- Services are loosely coupled
- Services are stateless
- Services are discoverable

www.mas-ict.ch

Hes·so

So… what is Service Orientation?

- Interoperability-centric approach to interconnect applications and facilitates complex interactions between heterogeneous and autonomous systems
- Paradigm shift from RPC-based/object-based architecture to a loosely-coupled, message-focused and service-oriented architecture
- SOA = {Services, Messages, QoS/Policy}



HumanTech
Technology for Human Wellbeing Institute

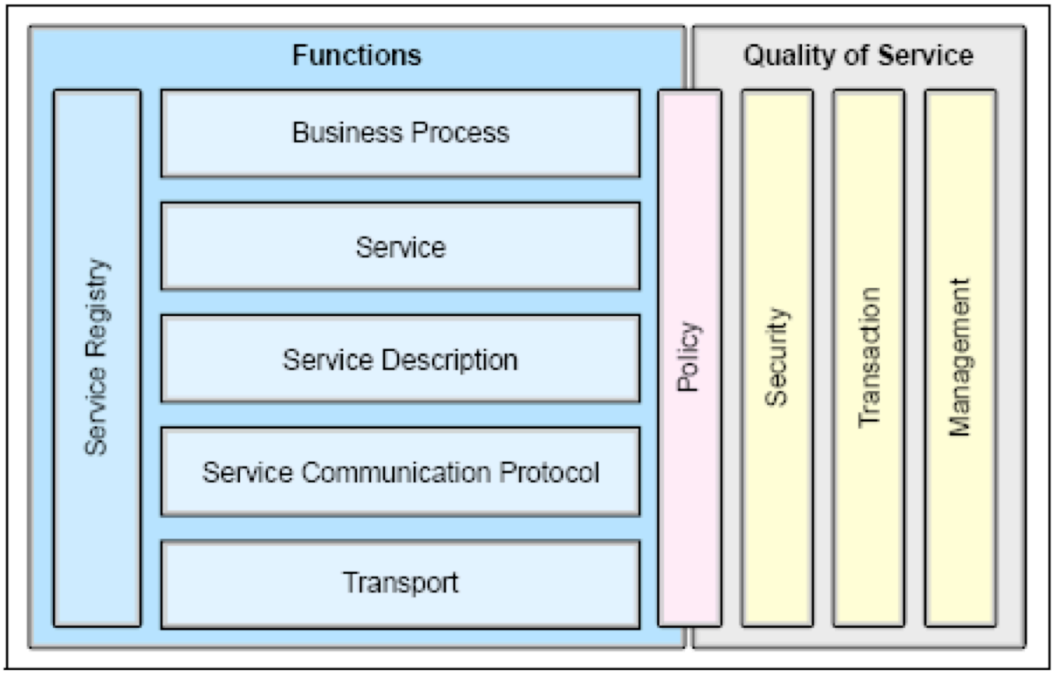
Building Service Oriented Applications is a complex undertaking, design patterns and frameworks have an important role to play to ease the process and to provide high Quality of Service (QoS) attributes.

Built around the concepts of service and message, a service is an entity that can send and receive message. *Service interaction is facilitated by exchanging messages*

•A service adheres to a contract: Describes the format of the messages exchanged. Defines the message exchange patterns in which a service is prepared to participate.

•Messages include: Application data, Control information (Addressing, Security, Reliability)

SOA Components



SOA Functional Components:

•**Transport** Mechanism used to move service requests from the service consumer to the service provider, and service responses from the service provider to the service consumer.

•**Service Communication Protocol** Mechanism the provider and consumer use to communicate what is being requested and what is being returned.

•**Service Description** An agreed schema for describing what the service is, how it should be invoked, and what data is required to invoke the service successfully.

•**Service** An actual service that is made available for use.

•**Business Process** Collection of services, invoked in a particular sequence with a particular set of rules, to meet a business requirement. Note that a business process could be considered a service in its own right, which leads to the idea that business processes may be composed of services of different granularities.

•**Service Registry** Repository of service and data descriptions which may be used by service providers to publish their services, and service consumers to discover or find available services. The service registry may provide other functions to services that require a centralized repository.

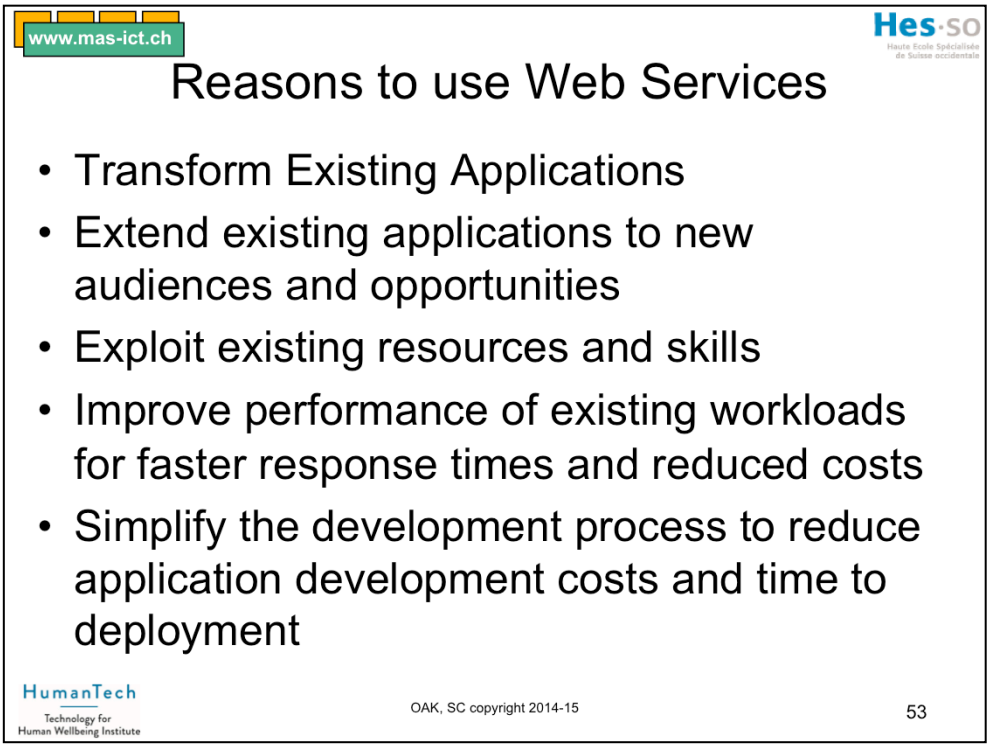
SOA QOS Components

•**Policy** A set of conditions or rules under which a service provider makes the service available to consumers. There are aspects of policy which are functional, and aspects which relate to quality of service; therefore we have the policy function in both functional and quality of service areas.

•**Security** The set of rules that might be applied to the identification, authorization, and access control of service consumers invoking services.

•**Transaction** The set of attributes that might be applied to a group of services to deliver a consistent result. For example, if a group of three services are to be used to complete a business function, all must complete or none must complete.

•**Management** The set of attributes that might be applied to managing the services provided or consumed.



Web Services -

Customers are looking to redefine their applications quickly and effectively to meet their business demands. There is a need for rapid business process adaptation and reshaping. Application maintenance consuming 60-80% of IT budgets and staff turnover or retirement lessens individual programmer familiarity with existing systems, application

maintenance efficiency is key driver.

There is also a need to meet increasing development workloads. The growth in complexity of development platforms and integration needs will force organizations to turn away from code-centric development practices in exchange for more efficient development paradigms. They need better tooling to deliver more effective and efficient development processes.

Industry adoption and proliferation of Web Services capabilities into development platforms and tools are making it easier for companies to adopt a service-based development approach. The need for richer than HTML experiences and disconnected operations will lead most companies to adopt multiple user interfaces delivery architectures.

Finally, Because of recent pressures for cost reductions and market demand for better processes, we expect continued pressure from business executive to switch to new, business-differentiating activities. There will be a continued strong drive from business for process improvements.

Web Services

- Web services provides a distributed computing approach for integrating extremely **heterogeneous** applications over the Internet.
- The Web service specifications are completely **independent** of programming language, operating system, and hardware to promote loose coupling between the service consumer and provider.
- Using open **standards** provides broad interoperability among different vendor solutions. This means that companies can implement Web services without having any knowledge of the service consumers, and vice versa, facilitating just-in-time integration and allows businesses to establish new partnership easily and dynamically.

Web Service Defined

A Web service is a software application identified by a URI, whose interfaces and bindings are capable of being defined, described, and discovered as XML artifacts. A Web service supports direct interactions with other software agents using XML-based messages exchanged via Internet-based protocols. Basic Web services combine the power of two ubiquitous technologies: XML, the universal data description language; and the HTTP transport protocol widely supported by browser and Web servers.

Web services = XML + transport protocol (such as HTTP)

SOAP WEB SERVICES

Web Services Technology

Web Services Technology Stack

Layer Description	Implementation(s)	Other Concerns
Standard Messaging	Electronic Business XML Initiative (ebXML)	Quality of Service, Management, Security, Service Development
Service Composition	Business Process Execution Service for Web Services (BPEL4WS)	
Service Registry	Universal Description, Discovery and Integration (UDDI) ebXML Registries	
Service Description	Web Services Description Language (WSDL)	
Service Messaging	Simple Object Access Protocol (SOAP)/Extensible Markup Language (XML)	
Service Transport	Hypertext Transfer Protocol (HTTP) Simple Mail Transfer Protocol (SMTP) File Transfer Protocol (FTP)	

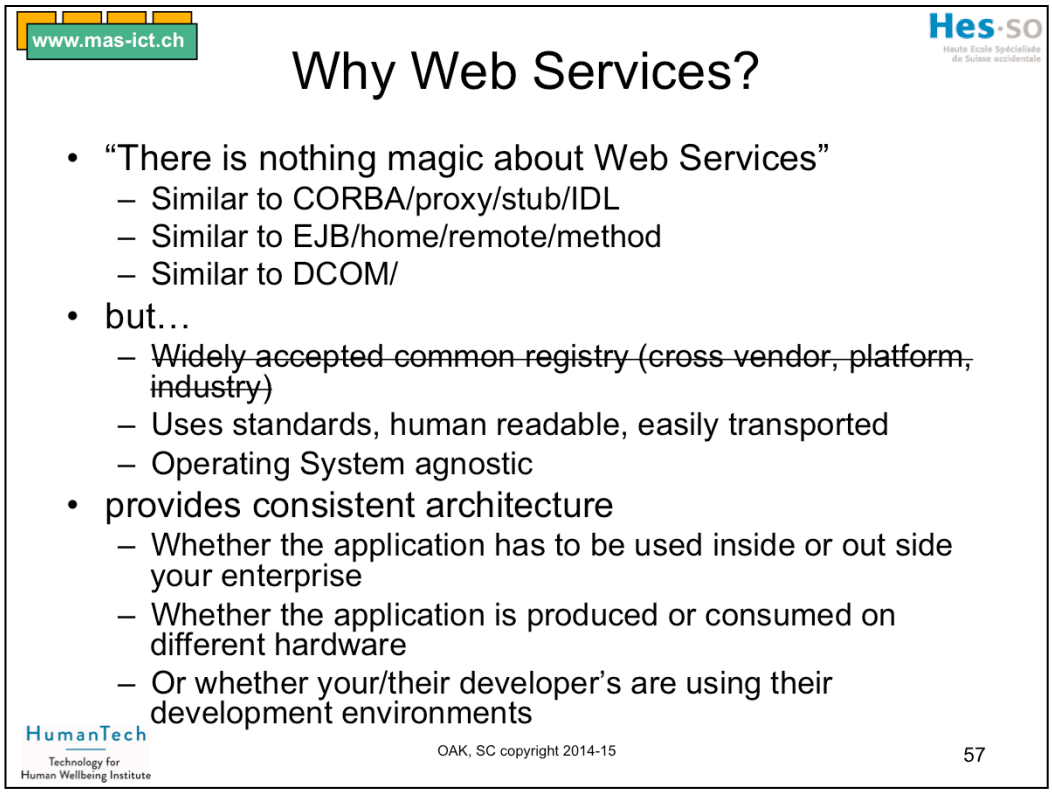
Web Services specifications:
•Registration and Discovery: Web services must be registered in universally accessible directory
•Service Description: Web Services interface requirementsand functionality must be described
•Messaging: Web Services must be able to communicate through corporate firewalls and over public networks
Standard Protocols: Functionality must be achievable using OPEN and freely available standards.

The technology is based on open technologies such as:

- Extensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)
- Universal Description, Discovery and Integration (UDDI)
- Web Services Description Language (WSDL)

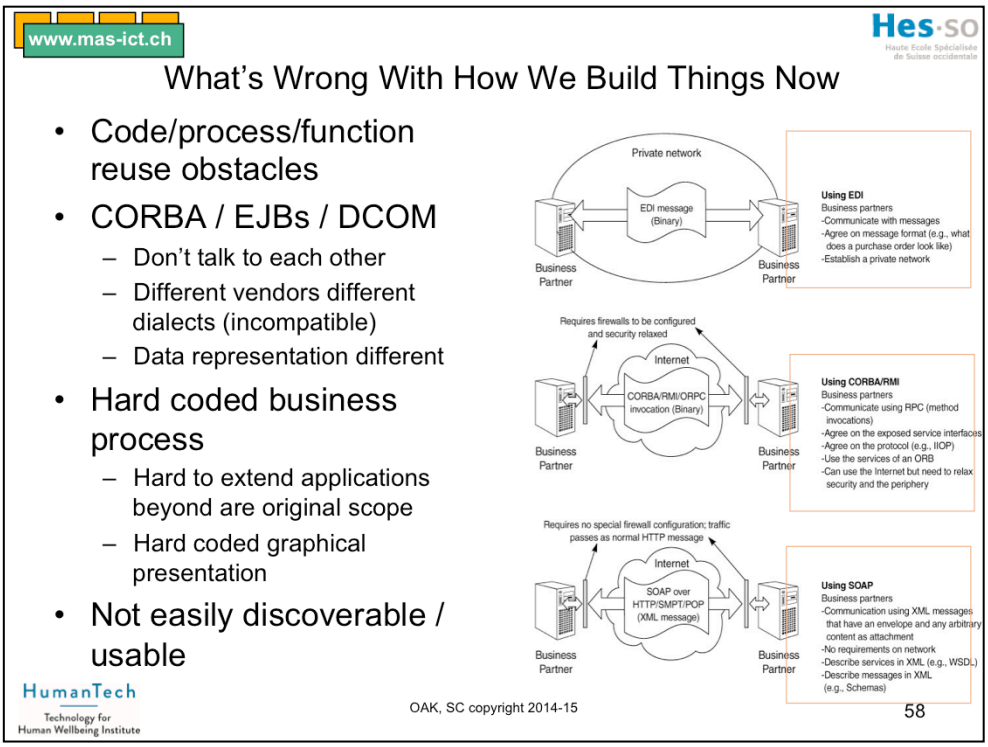
The core principles that have driven the design and implementation of the Web services architecture protocols are as follows:

1) Message orientation Using only messages to communicate between services and realizing that messages often have a life beyond a given transmission event.
2) Protocol composability Avoiding monoliths through the use of infrastructure protocol building blocks that can be used in nearly any combination.
3) Autonomous services Allowing endpoints to be independently built, deployed, managed, versioned, and secured.
4) Managed transparency Controlling which aspects of an endpoint are (and are not) visible to external services.
5) Protocol-based integration Restricting cross-application coupling to wire artifacts only.



Web Services: Key Features

1. **Web services are self-contained**: On the client side, no additional software is required. A programming language with XML and HTTP client support, for example, is enough to get you started. On the server side, merely a Web server and a servlet engine are required. It is possible to Web service enable an existing application without writing a single line of code.
2. **Web services are self-describing:** Neither the client nor the server knows or cares about anything besides the format and content of request and response messages (loosely coupled application integration). The definition of the message format travels with the message. No external metadata repositories or code generation tools are required.
3. **Web services are modular:** Web services are a technology for deploying and providing access to business functions over the Web; J2EE, CORBA, and other standards are technologies for implementing these Web services.
4. **Web services can be published, located, and invoked across the Web. The standards required to do so are:** Simple Object Access Protocol (SOAP), also known as service-oriented architecture protocol, an XML-based RPC and messaging protocol. Web Service Description Language (WSDL), a descriptive interface and protocol binding language. Universal Description, Discovery, and Integration (UDDI), a registry mechanism that can be used to look up Web service descriptions
5. **Web services are language independent and interoperable:** The interaction between a service provider and a service requester is designed to be completely platform and language independent. This interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP). Because the service provider and the service requester have no idea what platforms or languages the other is using, interoperability is a given.
6. **Web services are inherently open and standards based.** XML and HTTP are the technical foundation for Web services. A large part of the Web service technology has been built using open source projects. Therefore, vendor independence and interoperability are realistic goals.
7. **Web services are dynamic.** Dynamic e-business can become a reality using Web services because, with UDDI and WSDL, the Web service description and discovery can be automated.
8. **Web services are composable.** Simple Web services can be aggregated to create more complex services, either using workflow techniques or by calling lower-layer Web services from a Web service implementation.



Private network approach

The first approach takes the network for granted and concentrates on defining how parties communicate. This involved building wide-area networks and allowing the parties to plug into them. Often, these parties were valued business partners. This was the approach adopted by Electronic Data Interchange (EDI). EDI defined message constructs and how those messages were exchanged but left the network details to the parties involved. This was an acceptable solution for large corporations that had established relationships with their partners, and it was, in fact, quite successful. However, it resulted in collections of networks that were not only expensive but that also locked the parties into using proprietary products and solutions.

Communication protocol

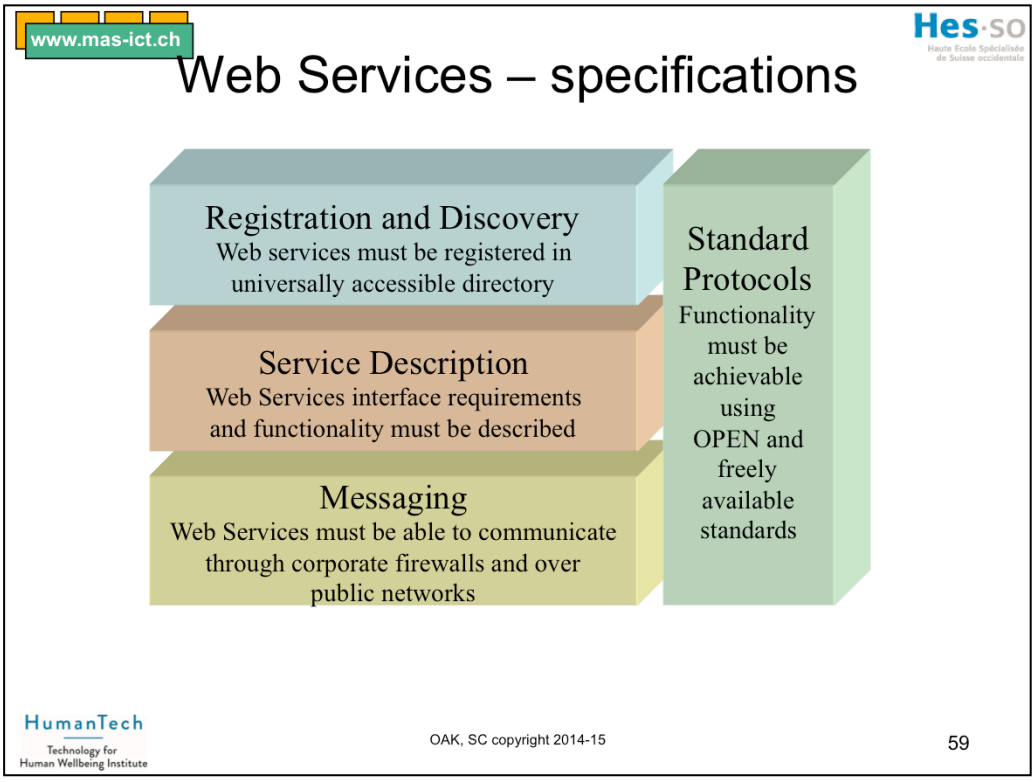
The second approach started with standardizing the communication protocol and involved building distributed computing infrastructures that could run on an open network. This was the route taken by CORBA, RMI, and DCOM. Each developed its own protocol (IIOP, JRMP, and ORPC, respectively) that was positioned on top of TCP/IP and allowed two distributed objects to communicate.

While this approach reduced the need for private networks, the problem still remained: CORBA implementations could communicate with CORBA, RMI with RMI (with RMI-IIOP in J2SE 1.2 RMI and CORBA), and DCOM with DCOM. Though CORBA promoted platform independence by allowing objects to be written in any language, it still required both parties to use underlying object request broker (ORB) software, which usually had to be from the same vendor, because of the fragmentation in vendor implementations when it came to interoperability.

SOAP

It is from this background that SOAP has been developed. SOAP decouples the data format and the underlying protocol, by leveraging a platform-independent language such as XML to describe the data and allowing for the use of well-established protocols such as HTTP to transport that data across the network. Some aspects that contribute to the simplicity of the SOAP protocol come from the fact that SOAP does not need to support features found in its predecessors, IIOP and JRMP, nor does it require the use of any broker software. For example, SOAP does not support distributed garbage collection, nor does it support the RMI/CORBA concept of servant activation.

Sources: « Java Web Services Architecture by James McGovern, Sameer Tyagi, Michael Stevens and Sunil Matthew ISBN:1558609008 »



Web Service Architecture:

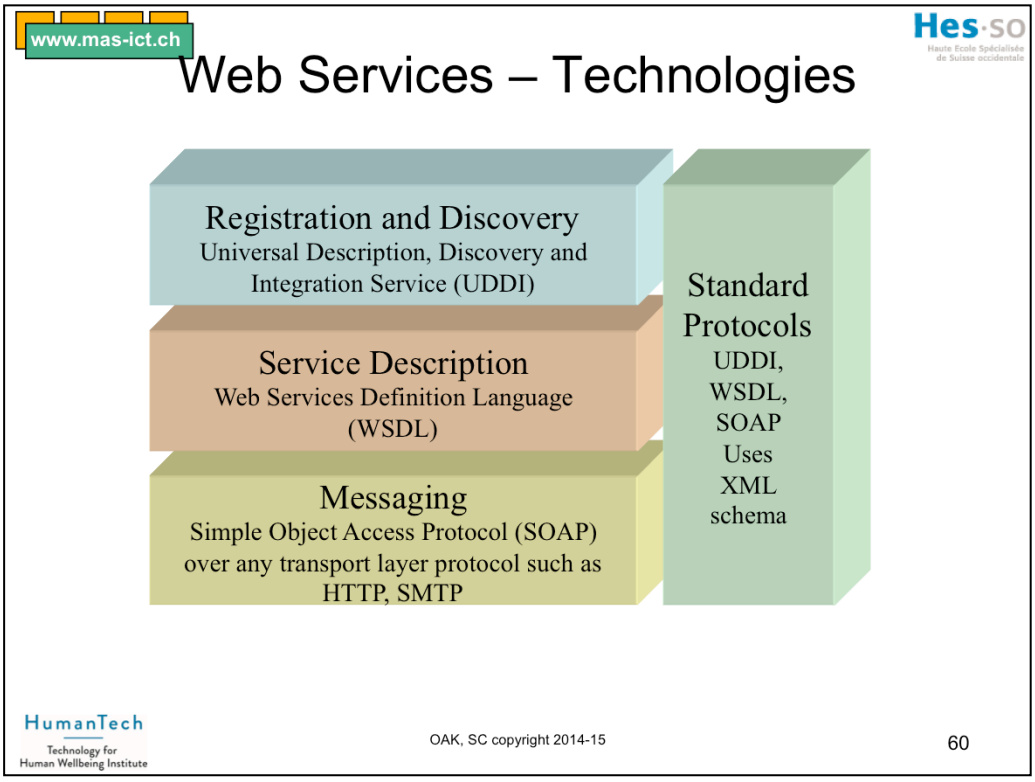
There are two ways to view the web service architecture. The first is to examine the individual roles of each web service actor; the second is to examine the emerging web service protocol stack.

Web Service Roles:

Service provider: This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service registry: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized clearinghouse for companies and their services.



Web Service Protocol Stack

Service transport

This layer is responsible for transporting messages between applications. Currently, this layer includes hypertext transfer protocol (HTTP), Simple Mail Transfer Protocol (SMTP), file transfer protocol (FTP).

XML messaging

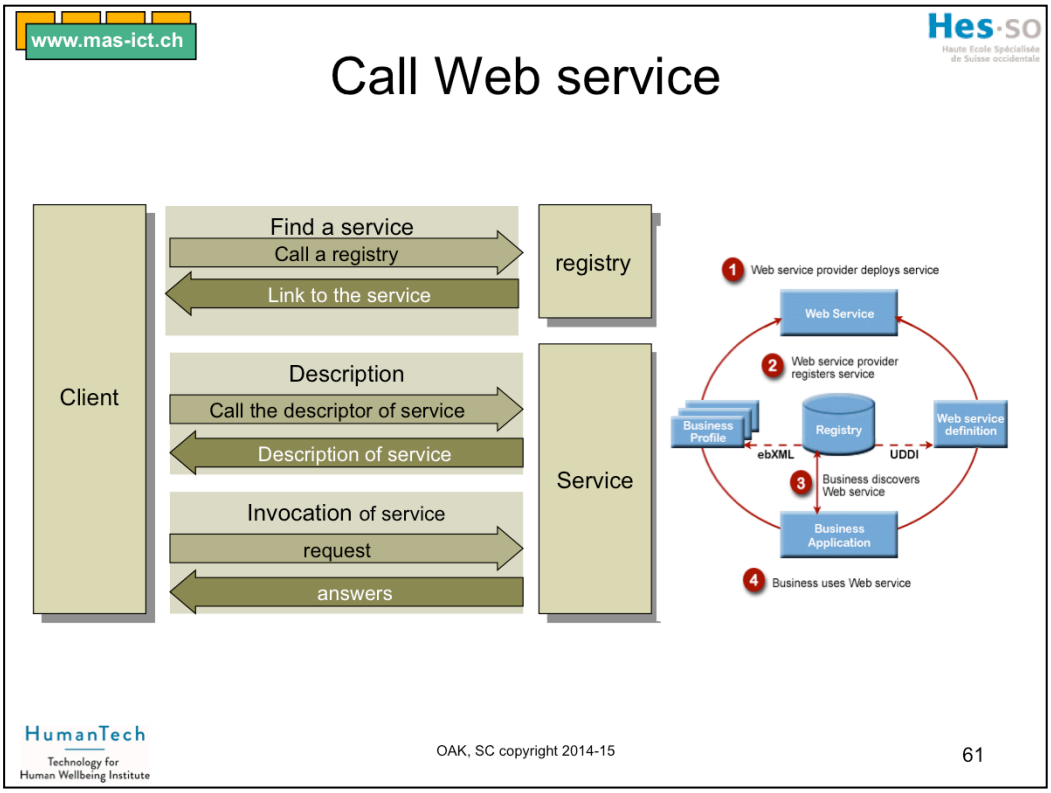
This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this layer includes XML-RPC and SOAP.

Service description

This layer is responsible for describing the public interface to a specific web service. Currently, service description is handled via the Web Service Description Language (WSDL).

Service discovery

This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via Universal Description, Discovery, and Integration (UDDI).

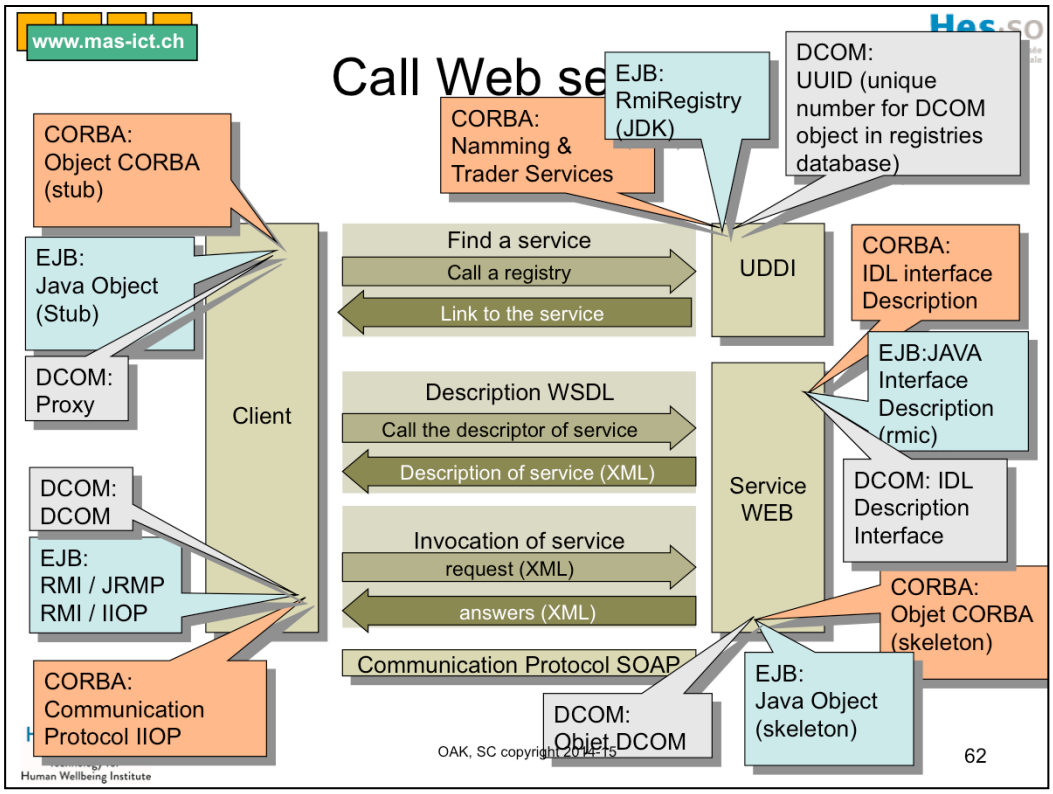


The Web services approach greatly simplifies B2B collaboration and provides a new model for the way businesses share their data and systems. In the Web services model, the Internet is leveraged to present a wide array of services, that is, self-contained modules of business function, that can be discovered and used on demand. As the following figure shows:

1) After creating and deploying one or more of its services ...
2) A business registers those services in a registry on the Internet. The registry contains information about the service and points to additional information.
3) Another business, through an application program, can then discover a needed service by searching the registry. The Web services model supports standard registries such as business registries that conform to UDDI or ebXML.
4) Using the information it discovers in a UDDI registry, a business application locates the appropriate server that runs the Web service. The application also locates a service definition that describes, among other things, how to make requests to the service. Using an ebXML registry involves the exchange of business profile information between the Web service provider and the Web service requester. After the exchange, the two parties agree on a business arrangement. The requester then locates and uses the Web service according to the business arrangement.

ebXML provides a comprehensive B2B framework for business collaboration in a global electronic marketplace.

Sources: http://java.sun.com/developer/technicalArticles/WebServices/WSPack/



What is CORBA?

The Common Object Request Broker Architecture (CORBA) is the Object Management Group's specification for achieving interoperability between distributed computing nodes. Their objective was to define an architecture that would allow heterogeneous environments to communicate at the object level regardless of who designed the two endpoints of the distributive application.

What is DCOM?

DCOM is the distributed extension of Microsoft's COM (Component Object Model) , which builds an object remote procedure call (ORPC) layer on top of RPC to support remote objects. A COM server can create object instances of multiple object classes. A COM object can support multiple interfaces, each representing a different view or behavior of the object. An interface consists of a set of functionally related methods. A COM client interacts with a COM object by acquiring a pointer to one of the object's interfaces and invoking methods through that pointer, as if the object resides in the client's address space. COM specifies that any interface must follow a standard memory layout, which is the same as the C++ virtual function table.

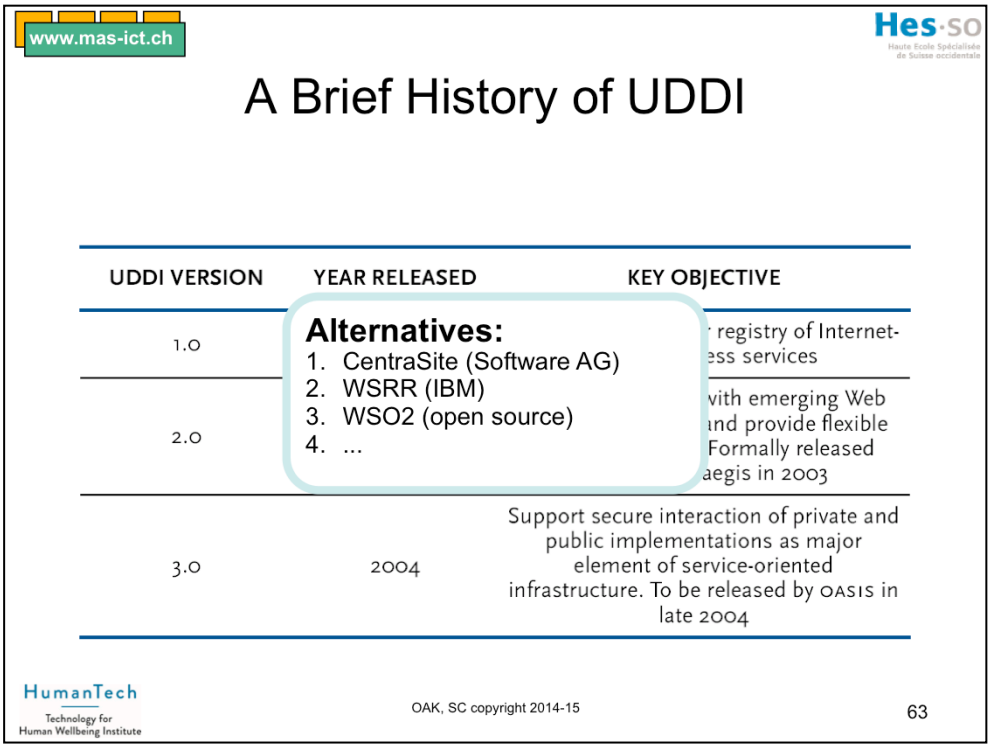
Client and Server components in different RPC architectures

RPC architectures | Client Stub | Server Stub

CORBA | Stub | Skeleton

DCOM | Proxy | Stub

Web services | Service Proxy | Service Implementation Template



Key Functional Concepts in the UDDI Specification

UDDI describes a registry of Web services and programmatic interfaces for publishing, retrieving, and managing information about services.

The UDDI specification defines services that support the description and discovery of

1. businesses, organizations, and other Web services providers,
2. the Web services they make available, and
3. the technical interfaces which may be used to access and manage those services.

UDDI is based upon several other established industry standards, including HTTP, XML, XML Schema, SOAP and WSDL.

…at version 3.0.2 since February 2005

Alternatives:

HP Systinet, IBM WSRR, Software AG CentraSite are commercial solutions.

Open source options include Galaxy and WSO2 Registry.

UDDI

- Understanding UDDI
  - The UDDI service registry provides a Web services directory platform. It is a centralized Web service search engine helping Web service consumer applications to find adequate service offerings.
- In an UDDI registry consumers may find:
  - Information about businesses and organizations offering Web services
  - Descriptions of the Web services that these organizations provide
  - Information about technical interfaces to these Web services

UDDI is the protocol for communicating with registries.

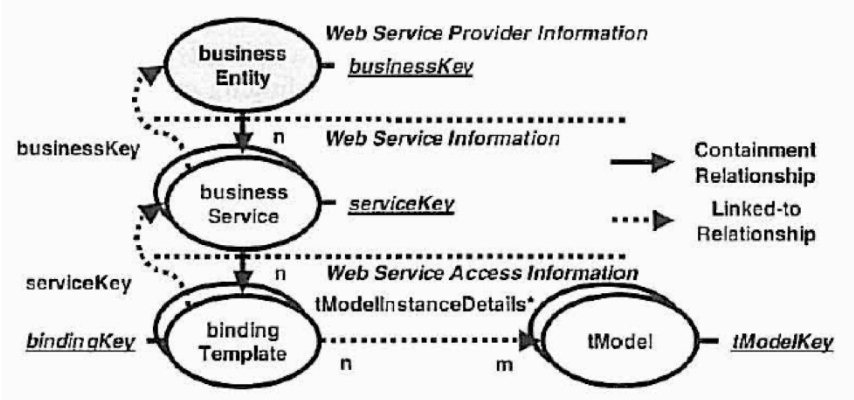
The Organization of UDDI

Directory	Operation	Information
White pages: Name, address, telephone number, and other contact information of a given business	**Publish:** How the provider of Web services registers itself	**Business information:** A businessEntity object contains information about services, categories, contacts, URLs, and other things necessary to interact with a given business.
Yellow pages: Categories of businesses based on existing (nonelectronic) standards	**Find:** How an application finds a particular Web service	**Service information:** Describes a group of Web services. These are contained in a businessService object.
Green pages: Technical information about the Web services provided by a given business	**Bind:** How an application connects to and interacts with Web services after it's been found	Binding information: The technical details necessary to invoke Web services. This includes URLs, information about method names, argument types, and so on. The bindingTemplate object represents this data.
		Service specification detail: This is metadata about the various specifications implemented by a given Web service. These are called tModels in the UDDI specification.

www.mas-ict.ch

Hes·so

UDDI data Structure



Information entity	Key attribute name
businessEntity	businessKey
businessService	serviceKey
bindingTemplate	bindingKey
tModel	tModelKey

Source : Perspectives on Web Services, Olaf Zimmermann, Springer, ISBN: 3-540-00914-0, 2003

The UDDI Registry Structure:

The UDDI service registry data model is composed of five main data types. Which are defined through XML schemas. They are organized hierarchically and provide two types of information, namely business-related information about a Web service provider and the Web services it offers and technical information about Web service bindings. The data structures are:

business entity: The business entity data structure externalizes business level information about a Web service provider, such as the name of a business, a business contact or a general categorization of the business. It serves as the top object of the containment hierarchy in the UDDI registry.

business service: A service provider identified through the business entity may offer several types, or collections, of related Web services. Each aggregation of correlated Web services is represented through a business service data structure. It provides overall Web service-level information, such as the name of a Web service aggregate, a description, or a service categorization. The information is of a non-technical nature.

binding template: The binding template data structure exposes a service endpoint address required for accessing a distinct Web service from a technical point of view. It may also describe technical characteristics of a service implementation or refer to remotely hosted services. A business service potentially contains multiple binding templates; one for each Web service.

tModel: A tModel (or "technical model") data structure exposes Web service interface information. A Web service may, for example, be compliant to distinct specifications or standards or it may follow particular conventions. A client that is not aware of these standards or conventions can fail to interact properly with the service. Hence, tModel data provides a technical specification, or fingerprint, that helps a consumer to find out whether its capabilities meet the technical prerequisites for service invocation. A tModel may also represent a value system helping to identify and categorize information entities.

The Identification of Information Entities: Individual UDDI information entities must be uniquely identified through the value of a key attribute. Such a key attribute is present for each entity.

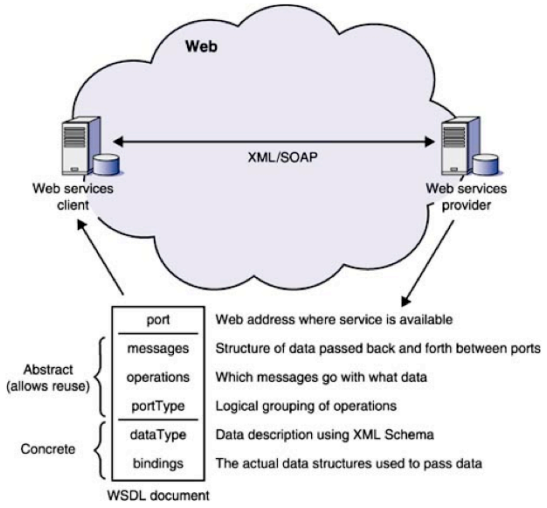
WSDL

- Understanding WSDL
  - Consumer applications automatically binding to services need a formal service description that provides information about service locations and service interfaces.
    - With this information, the consumer knows how to interact with a service.
  - The Web Services Description Language (WSDL) is a formal language specification for providing this type of information.
    - A WSDL document is basically a service interface contract, written in XML, to which the consumer must agree. It may read several of these contracts in order to find the interaction conditions that best meet its requirements.

The WSDL Building Blocks

- WSDL separates a service description into two parts:
  - abstract interface
    - The interface description unfolds the general Web service interface layout
      - the operations supported by the service
      - the operation parameters and abstract data types.
    - This description is completely independent of any concrete network address, communication protocol or concrete data structures.
  - Concrete implementation
    - The implementation description binds the abstract interface description to a concrete network address, to a protocol and to concrete data structures. A Web service consumer may bind to such an implementation and invoke a service.

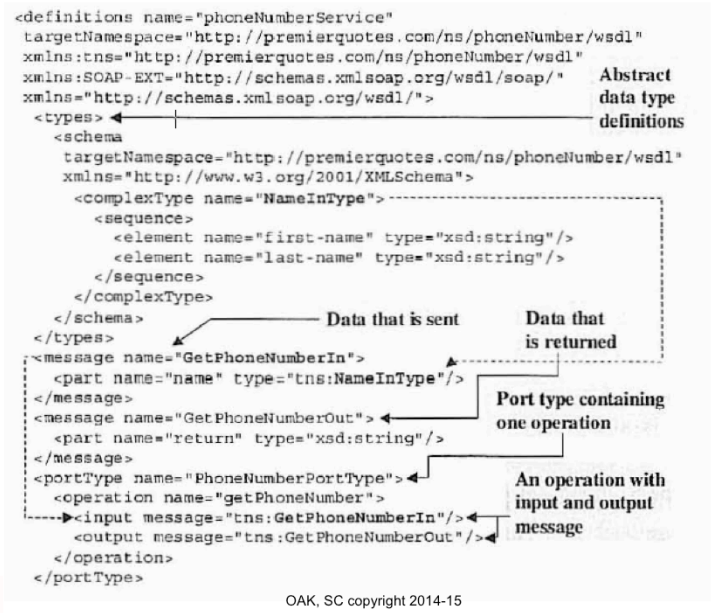
The structure of a WSDL document



WSDL

Web Services Description Language

The WSDL Interface Description




```
<definitions name="phoneNumberService"
 targetNamespace="http://premierquotes.com/ns/phoneNumber/wsdl"
 xmlns:tns="http://premierquotes.com/ns/phoneNumber/wsdl"
 xmlns:SOAP-EXT="http://schemas.xmlsoap.org/wsdl/soap/"
 xmlns="http://schemas.xmlsoap.org/wsdl/">
  <types>
    <schema
     targetNamespace="http://premierquotes.com/ns/phoneNumber/wsdl"
     xmlns="http://www.w3.org/2001/XMLSchema">
      <complexType name="NameInType">
        <sequence>
          <element name="first-name" type="xsd:string"/>
          <element name="last-name" type="xsd:string"/>
        </sequence>
      </complexType>
    </schema>
  </types>
  <message name="GetPhoneNumberIn">
    <part name="name" type="tns:NameInType"/>
  </message>
  <message name="GetPhoneNumberOut">
    <part name="return" type="xsd:string"/>
  </message>
  <portType name="PhoneNumberPortType">
    <operation name="getPhoneNumber">
      <input message="tns:GetPhoneNumberIn"/>
      <output message="tns:GetPhoneNumberOut"/>
    </operation>
  </portType>
```

HumanTech
Technology for Human Wellbeing Institute

The central element externalizing an interface description is the portType element. The port type contains all operations supported by a Web service. In Figure, the port type named PhoneNumberPortType only supports one operation. which is called getPhoneNumber. As you can see, operations are likewise represented via XML elements. Assuming the Web service offered several operations, they would all appear as child elements underneath the portType element.

The getPhoneNumber operation is a request-response operation type, as it contains an input message and an output message. Other types of operations exist as well, such as for example a one-way operation. An operation holds all messages that are potentially exchanged between a Web service consumer and a service provider.

Message GetPhoneNumberIn is linked by name to the input element of the getPhoneNumber operation. Hence, the message represents the data that is sent from a consumer to a service provider. Message GetPhoneNumberOut is linked to the output message of the getPhoneNumber operation. This message encapsulates The data of the return value.

A message consists of parts, and parts are linked to types. While a message represents the overall data type set of an operation, parts may structure this set. In Figure the input message named GetPhoneNumberIn carries only one part that refers to a complex type, consisting of the first-name and the last-name elements.

The types element is a container that holds all abstract data types needed by the interface description.

Here is a summary of the elements collectively describing a Web service interface:

•portType: A portType is a named collection of operations

•Operation: An operation abstractly describes a service call. It may contain an input message, an output message and optionally several fault messages.

•Message: A message is an abstract description of the data that is send. Messages consist of logical units called parts.

•Part: Each part is associated with a data type.

•Types: The types element provides a container for data type definitions.

The WSDL Implementation Description

```
<binding name="PhoneNumberSoapBinding" type="tns:PhoneNumberPortType">
  <SOAP-EXT:binding style="rpc"
   transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="getPhoneNumber">
    <SOAP-EXT:operation soapAction=""/>
    <input>
      <SOAP-EXT:body use="encoded"/>
    </input>
    <output>
      <SOAP-EXT:body use="encoded"/>
    </output>
  </operation>
</binding>
<service name="PhoneNumberService">
  <port name="PhoneNumberPort" binding="tns:PhoneNumberSoapBinding">
    <SOAP-EXT:address
     location="http://www.premierquotes.com/servlet/rpcrouter"/>
  </port>
</service>
</definitions>
```

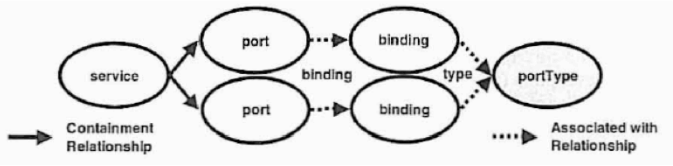
Use SOAP RPC style

Bind an abstract operation to this concrete implementation ...

... and map the abstract input and output messages to these concrete messages

Service name

Network address of the service



The implementation description provides a SOAP binding for the interface:

-The central element of the implementation description is the binding element. A binding maps the port type, the service interface description, to an existing service implementation. It provides information about the protocol and the concrete data formats expected by a service offered at a distinct network address.

-The binding element in Figure links the port type named PhoneNumberPortType (see Figure in previous slide for the port type definition) to the port named PhoneNumberPort. This happens through the binding name PhoneNumberSoapBinding, as you can see when following the dotted arrow. Several bindings may represent various implementations of the same port type.

The abstract operation GetPhoneNumber together with its input and output messages from the service interface description (see Figure in previous slide) is mapped to concrete SOAP messages. The data types belonging to these messages, which are abstractly described via XML schema in the interface description, should be SOAP encoded for transfer.

A port element describes a service network address. A service exchanges messages in a defined format through a port. A Web service consumer may bind to a port and interact with the service, provided that it understands and respects the concrete message format expected by the service. The same service may be offered with various data formats on multiple ports. A consumer that wants to interact with the service may choose one of these ports.

A service element finally combines all ports, i.e., all network addresses, at which a service is made available. The service element of a WSDL document may be the starting point for a consumer for exploring a service description. The separation between service and port promotes flexibility, as a Web service consumer binding to a service may always retrieve dynamically the appropriate network access point of the service.

•Binding: A binding provides concrete protocol and data format specifications for a particular port type.

•Port: A port describes me network address of a service. In combination with a binding, it provides information about a single concrete service endpoint.

•Service: A service is a collection of related ports. These ports may share the same port type, but employ different bindings or network addresses. They may also be concrete service endpoint implementations of several port types.

The Containment Structure of a WSDL Document

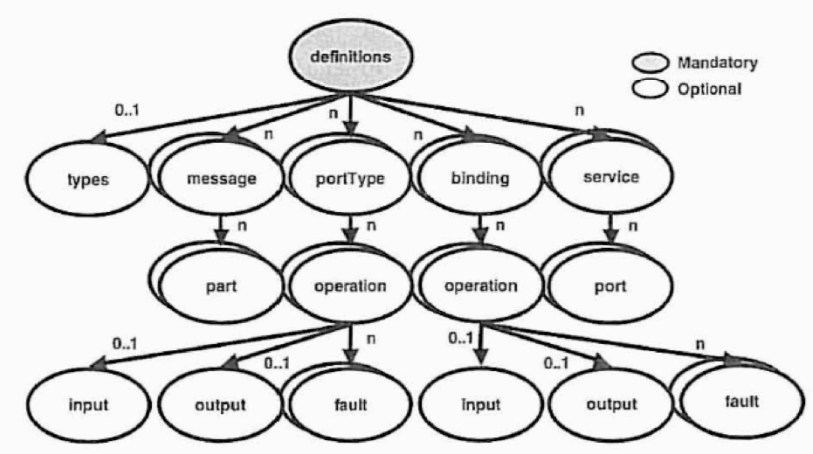
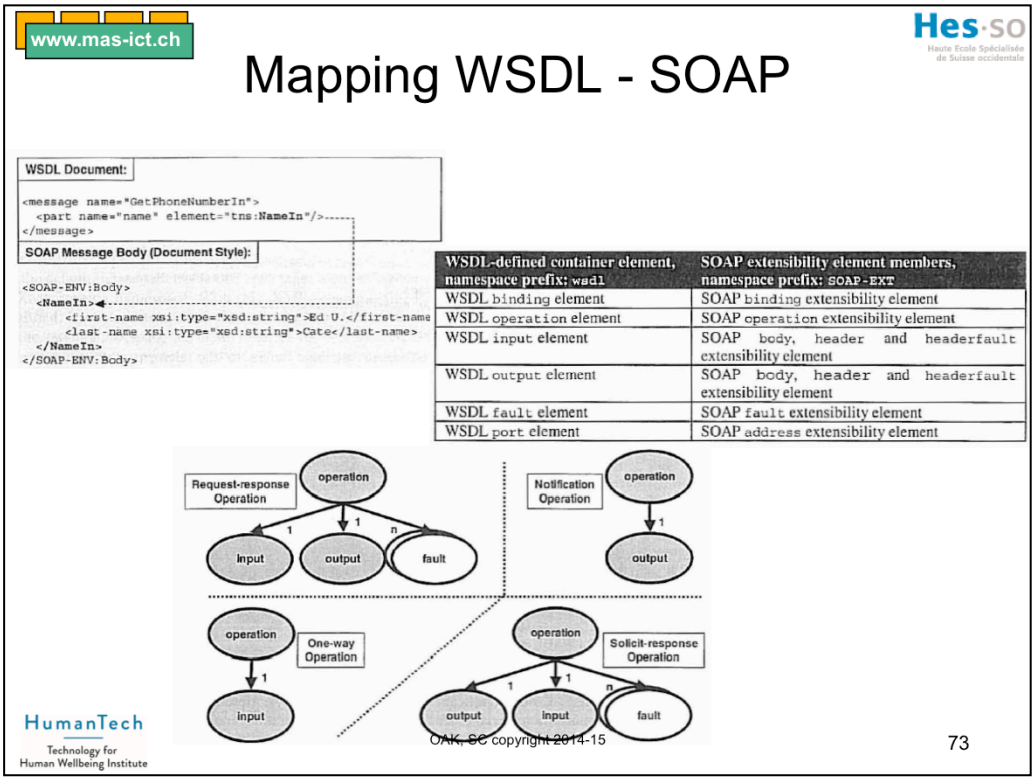


Figure may bring some more light onto the containment structure of a WSDL document. It also introduces the definitions element, which is a container for all WSDL elements. The figure visualizes how the abstract port type containment structure has an equivalent binding containment structure. The containment tree underneath the portType element looks identical to the containment tree beneath the binding element. Each binding related to a distinct port type maps a concrete protocol implementation to this port type.



Mapping of a WSDL part to a SOAP document style message. Hence, the WSDL document ultimately defines the appearance of SOAP messages. This should not be surprising, as the WSDL document describes the contract for service invocation.

The operation element: WSDL differentiates four operation types that a service endpoint can support. They are:

One-way: The endpoint receives a message. A one-way operation has only an input element specifying the operation message.

Request-response: The endpoint receives a message and sends a correlated message. A request-response operation contains one input element, one output element and optionally one or more fault elements. The ordering of these elements is significant. as it defines an operation to be request-response.

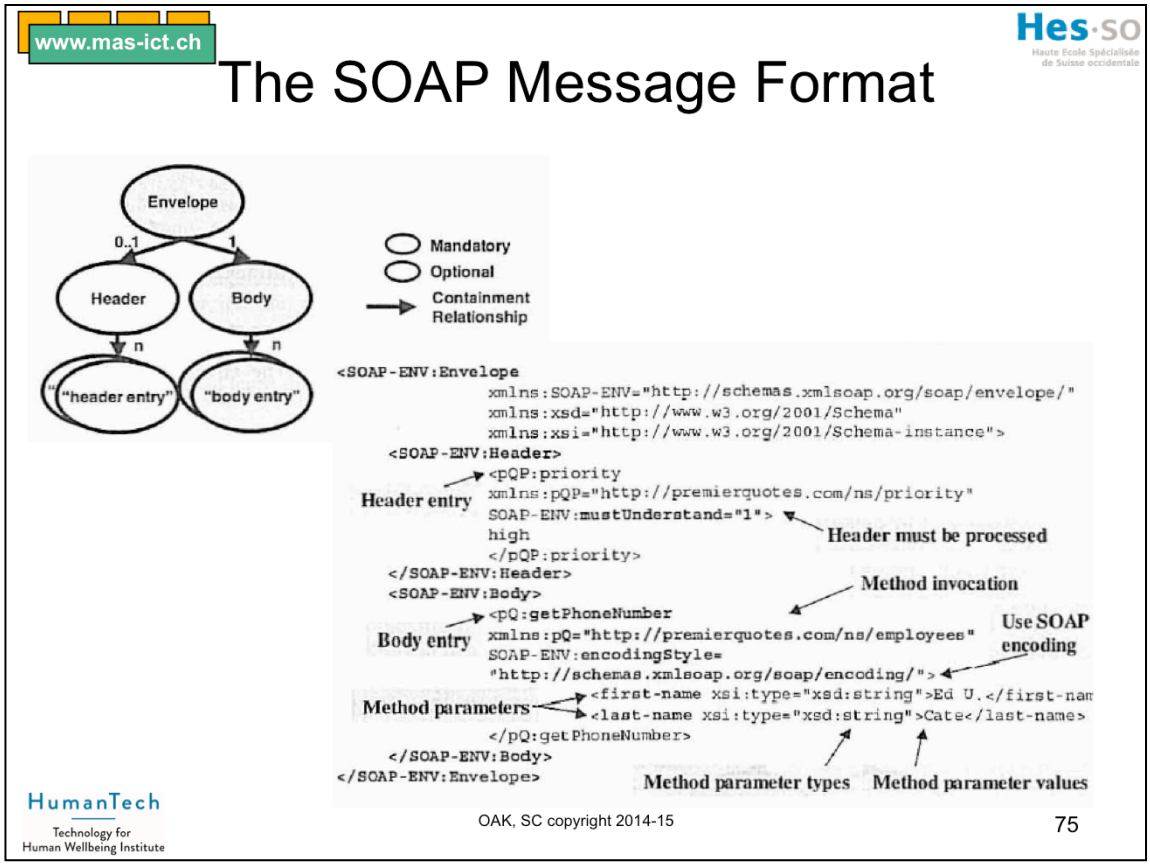
Solicit-response : The endpoint sends a message and receives a correlated message. A solicit-response operation contains one output element, one input element and optionally one or more fault elements. Again, ordering of the elements matters.

Notification: The endpoint sends a message. A notification operation contains only an output element specifying the operation message.

SOAP

- Understanding SOAP
  - SOAP is an XML-based protocol for exchanging structured and typed data between applications. In a Web services environment.
  - it enables communication between service requestors, service providers and discovery agencies.
  - SOAP is intended to be simple and lightweight.
  - It defines neither application semantics nor does it define or require a distinct programming model.

It also does not enforce any specific communication pattern. Therefore, a SOAP protocol data unit, or SOAP message, can be embedded into various transport protocols. HTTP initially was the preferred transport protocol for SOAP messages, but nothing prevents these messages from being transferred via a Message-Oriented Middleware (MOM) or other transport protocols.



Envelope: The envelope represents the SOAP message and provides a container for header and body.

Header: The header offers a way to pass any kind of additional processing or control information to recipients of a SOAP message, separate from the mandatory message body. For instance, headers could convey authentication or transaction control-related information, as well as quality of service, service billing and accounting-related data. All immediate child elements of the header are called *header entries.*

Body: The body carries all mandatory information for the final recipient of the message. All immediate child elements of the body are called *body entries.*

The envelope declares three namespace prefixes:

• Prefix SOAP-ENV identifies SOAP-defined elements, namely the Envelope, the Header, the Body, which we know already, and attributes mustUnderstand and encodingStyle. We talk about the meaning of these attributes in a moment.

• Prefix xsd refers to XML schema elements, namely the schema built-in type string.

• Prefix xsi appears in front of the XML schema instance type attribute, asserting the type of the first-name and last-name elements to be an XML schema string, respectively.

The Header element contains one header entry element having the local name priority and the namespace name http://premierquotes .com/ns/priority. The value of priority is high. The mustUnderstand attribute value '1' tells the Web service provider that it must understand the semantics of the header entry and that it must process the header. The Web service requestor demands express service delivery. The Body element encapsulates the service method invocation information, namely the method name getPhoneNumber, the method parameters first-name and last-name, and the associated data types and parameter values. Apparently the requestor is interested in the phone number of Ed U. Cale.

You can see in the SOAP body that the <*Envelope*> element is prefixed by the *soap:* namespace. This is the default used by .NET. The <*Envelope*> element contains the <*Body*> element, and inside the body is our CalculatePercent document. The document contains three elements, two of which represent the parameters supplied in the Web method, <*Percent*> and <*Number*>. These are both enclosed by the <*CalcPercent*> element, which represents our method.

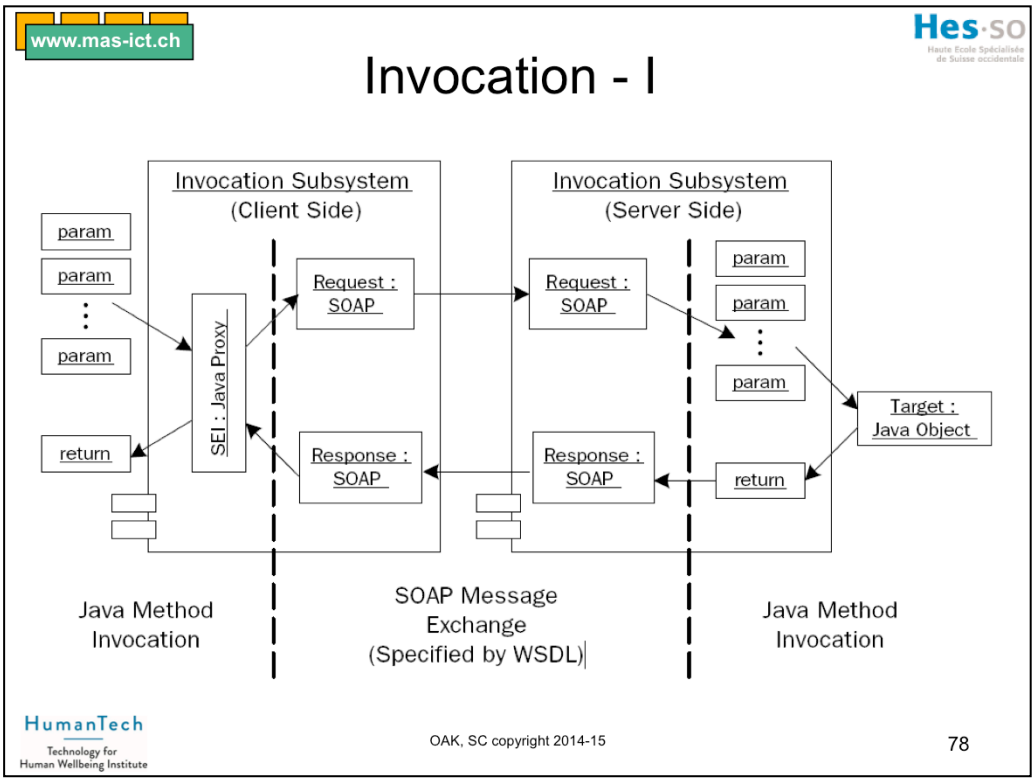
Note that both *Percent* and *Number* are actually strings in the SOAP. This string data is translated from the XML document into method parameters, which map to particular data types. Nothing in the SOAP provided here can stop textual data being passed, but the code would of course break when it got to the Web method. This translation is handled transparently by the code.

The SOAP Body

A successful response document to our previous request looks like the following:

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: length
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-
   instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <CalcPercentResponse xmlns="http://
   stefanocarrino.tsi.ch/">
      <CalcPercentResult>10</CalcPercentResult>
    </CalcPercentResponse>
  </soap:Body>
</soap:Envelope>
```


Just as our Web method returns only one value as a response, the SOAP body now has only one element to contain the response. This element, <*CalcPercentResult*>, contains the corresponding response to our request



There are invocation mechanisms on both the server side and the client side.

Server-Side Invocation

1. Receiving a SOAP message from a transport (e.g., from an HTTP or JMS endpoint).

2. Invoking handlers that preprocess the message (e.g., to persist the message for reliability purposes, or process SOAP headers).

3. Determining the message's target service—in other words, which WSDL operation the message is intended to invoke.

4. Given the target WSDL operation, determining which Java class/method to invoke. I call this the Java target. Determining the Java target is referred to as *dispatching.*

5. Handing off the SOAP message to the Serialization subsystem to deserialize it into Java objects that can be passed to the Java target as parameters.

6. Invoking the Java target using the parameters generated by the Serialization subsystem and getting the Java object returned by the target

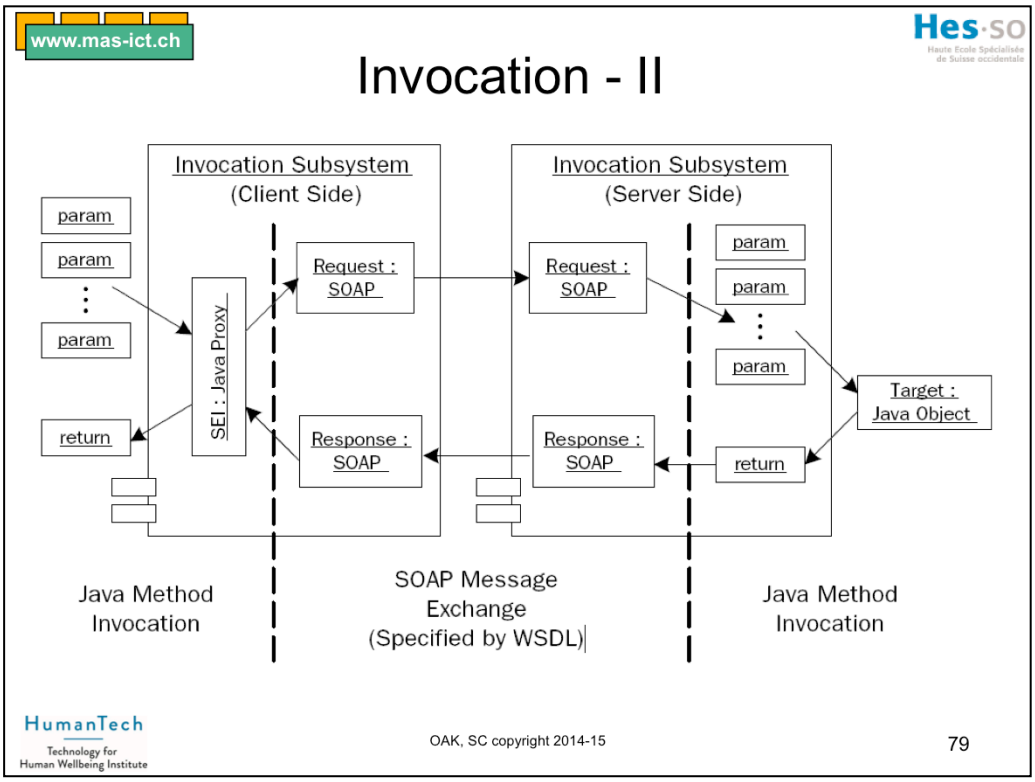
method.

7. Handing off the returned object to the Serialization subsystem to serialize it into an XML element conformant with the return message

specified by the target WSDL operation.

8. Wrapping the returned XML element as a SOAP message response conforming to the target WSDL operation.

9. Handing the SOAP response back to the transport for delivery.



On the client side, the invocation process is similar if you want to invoke a Web service using a Java interface.

Client-Side Invocation

1. Creating an instance of the Web service endpoint implementing a Java interface referred to (JWS terminology) as the service endpoint interface (SEI). The invocation subsystem has one or more factories for creating SEI instances. These instances are either created on the fly, or accessed using JNDI. Typically, SEI instances are implemented using Java proxies and invocation handlers.

2. Handling an invocation of the SEI instance.

3. Taking the parameters passed to the SEI and passing them to the Serialization subsystem to be serialized into XML elements that conform to the XML Schema specified by the target service's WSDL.

4. Based on the target service's WSDL, wrapping the parameter elements in a SOAP message.

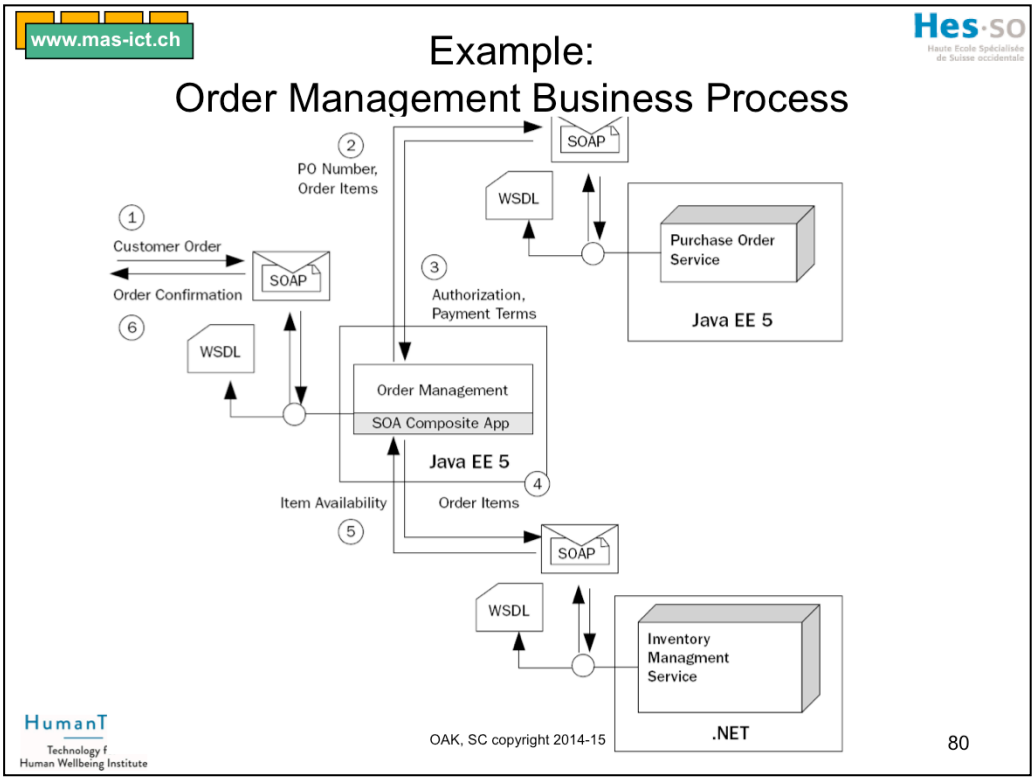
5. Invoking handlers that post-process the message (e.g., to persist the message for reliability purposes, or set SOAP headers) based on Quality of Service (QoS) or other requirements.

6. Handing off the message to the transport for delivery to the target Web service.

7. Receiving the SOAP message response from the transport.

8. Handing off the SOAP message to the Serialization subsystem to deserialize it into a Java object that is an instance of the class specified by the SEI's return type.

9. Completing the invocation



1. A customer sends a Customer Order containing, among other information, a PO (Purchase Order) Number and a list of Order Items—the products and quantities being ordered.

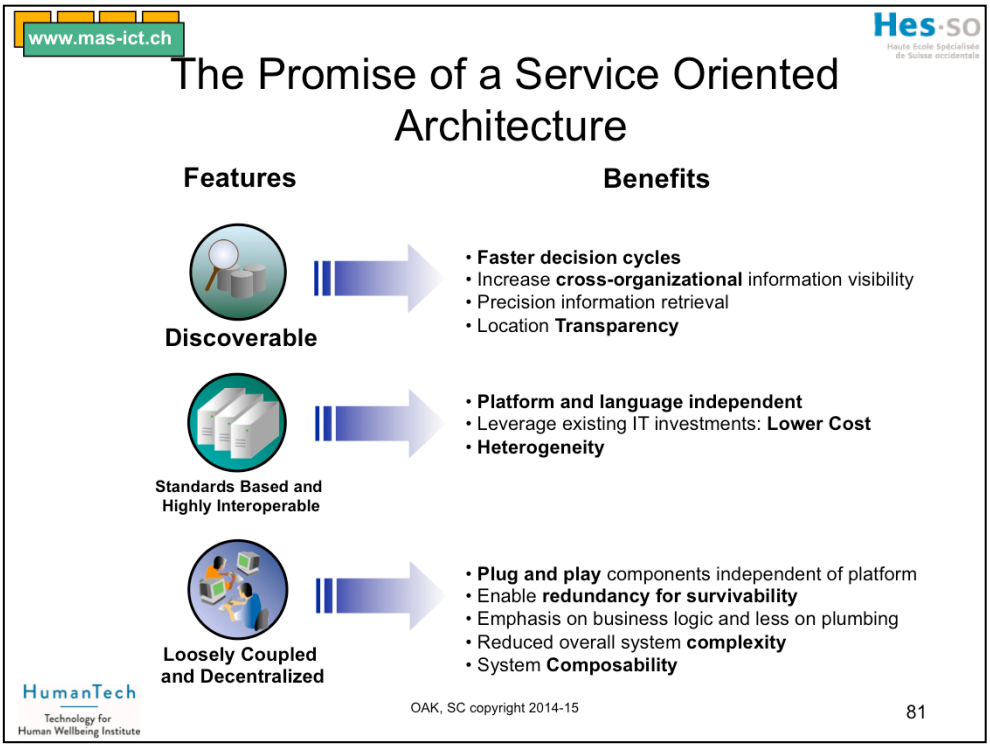
2. The PO Number and Order Items are passed to the Purchase Order System. It checks the Purchase Order to determine whether it covers the items being ordered and what payment terms are required.

3. If the PO covers the Order Items, an Authorization and a description of the required Payment Terms are returned.

4. Next, the Order Items are forwarded to the Inventory Management System to determine whether they are in stock and what the likely delivery dates are.

5. This information—the Item Availability—is returned.

6. Lastly, a response message is sent back to the customer—the Order Confirmation—detailing the payment terms and the anticipated delivery dates for the items.



SOA Benefits

•Leverages Existing IT Assets: Wrap Existing Applications As Services

•Simplifies Integration: Apps Integrate by having their services call other services, removes dependence on implementation. Becomes more important when integrating apps between companies.

•Enables Greater Responsiveness & Faster Time-To-Market :Reuse enables rapid App development & reduces lifecycle process times.

•Reduces Cost and Increases Revenue: Loose Coupling among core business services enables the quick introduction of new product/service support and lowers cost of doing so.

•Enables the Flexibility & Agility Needed To Respond To Future Conditions: Above factors enable a business to be more responsive to changing conditions.

SOA Benefits - The Overall Picture

Examples of Key Drivers	How can SOA help (not ordered)?
1. **Reduce IT costs** 2. **Support business agility and growth** 3. **Best Integration** 4. **Manage Supply Chain** 5. **Manage IT outsourcing** 6. **Support divisional business models**	• **Lower workload**: 1. Pre-configured business content 2. Lower integration effort (apps + partners) 3. Outsourcing of standard processes/services • **Lower costs**: 1. Infrastructure 2. Integration 3. Reusable assets • **Flexibility**: 1. Less complex applications (less code/bugs) 2. New services easy to develop/integrate 3. Services easy to adapt to business process change • **Process Automation**: 1. Workflows across applications 2. Automation of manual processes 3. Process integration with business partner

Web Services & SOA

- SOAP Web services are a technology that is well suited to implementing a service-oriented architecture. In essence, Web services are self-describing and modular applications that expose business logic as services that can be published, discovered, and invoked over the Internet. Based on XML standards, Web services can be developed as loosely coupled application components using any programming language, any protocol, or any platform. This facilitates the delivery of business applications as a service accessible to anyone, anytime, at any location, and using any platform.

Conclusion