

Mobile Web Services

Course ID: 582496

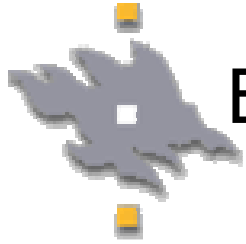
31 October 2005 - 08 December 2005

Monday & Thursday : 16:00-18:00

Web Service Architectures

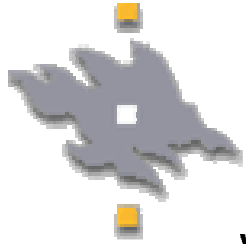
10 November 2005

Suresh Chande
Department of Computer Science
email: chande@cs.helsinki.fi



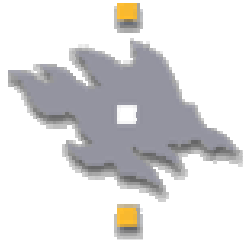
Basic Architectural components of Web Services Architectures

- Data Encapsulation -> XML
- Data typing and Validation -> XML Schema
- Messages and Message exchange -> SOAP
- Service Description -> WSDL
- Service description -> UDDI
- Web Services Development Models
 - Top-Down
 - Bottom-Up

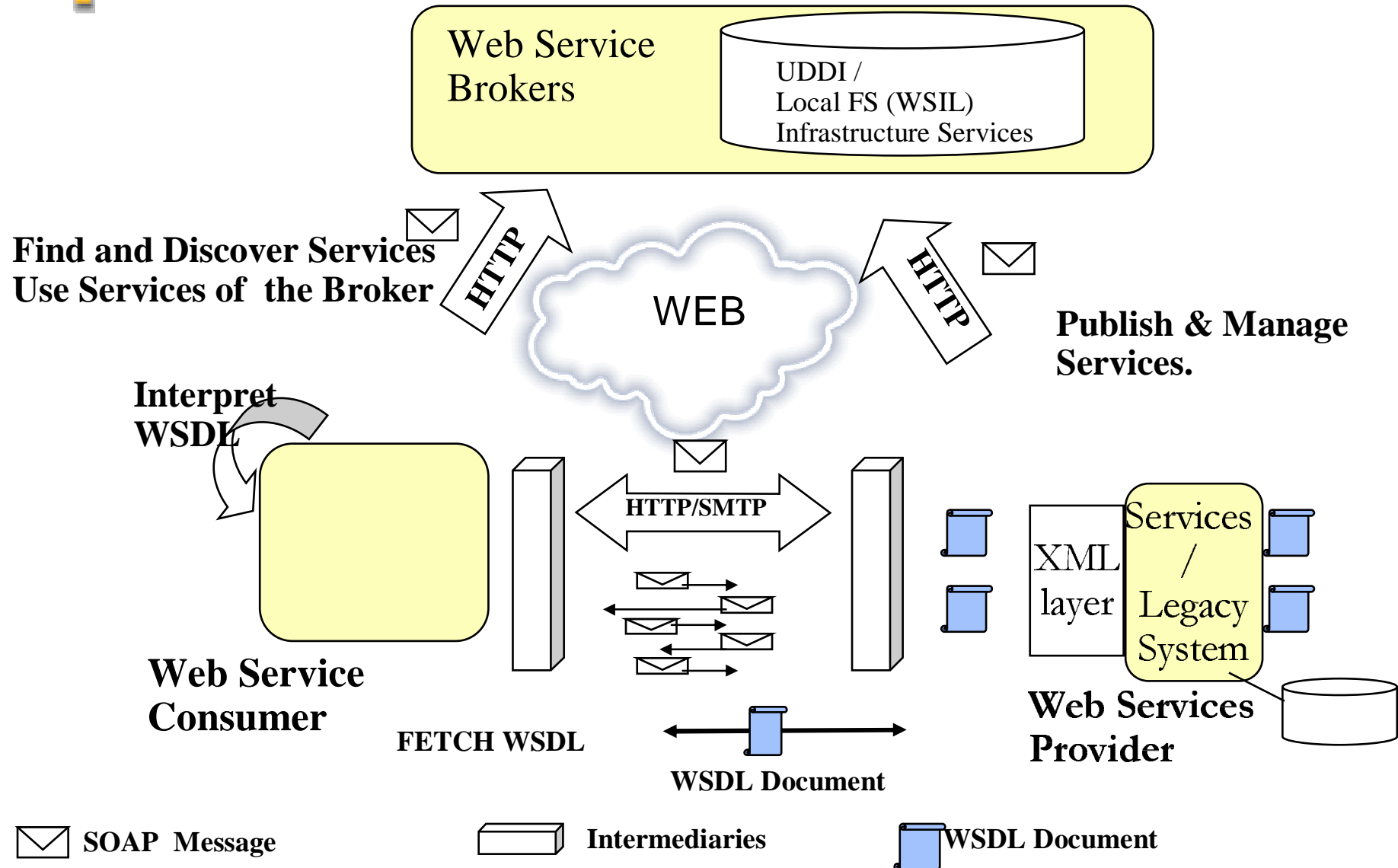


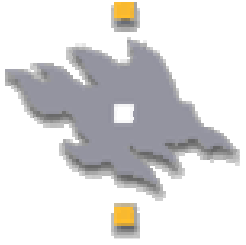
Web Services Architecture

- Web Service Architecture is defined around three main roles for its Web Services Components
 - **Web Services Client (WSC):** The Web Service(WC) consuming client system.
 - **Web Services Providers (WSP):** are participants who desire to publish their service as a Web Service
 - **Web Services Brokers (WSB):** Provide WSC and the WSP value added services such as: Discovery mechanism to find the required service, Publishing mechanism for the WSP means of publish and manage the description about the hosted WS.
 - WSB also provides many other infrastructure related services as Web Services for the WSC & WSP such as Authorization etc..



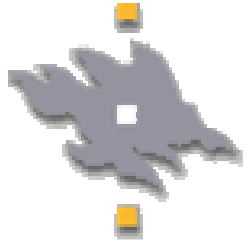
Web Services Architecture





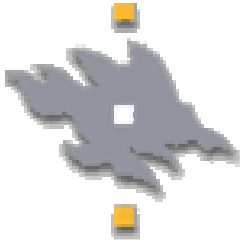
XML Schema





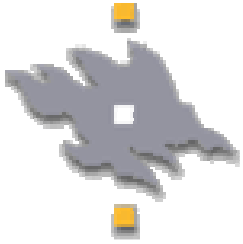
XML Schema – In A Nutshell

- XML Schema provides a method to define the structure, content and semantics of XML documents.
- An XML document which complies to a Scheme defined by an XML Schema is called an Instance Document.
- XML Schemas are Namespace aware and hence can leverage reuse of schema definitions
- XML schema is defined in terms of following two types:
 - **Simple Types:** A Simple XML elements which do not have sub elements/attributes: simple data types, enumeration, lists, restricted formatted values(patterns/regular expression evaluated)
 - **Complex Types:** An XML Element which could contain 1 or more sub-elements and attributes.
 - This also defines the sequence, Choice of occurrence, number of occurrence (min/max), default values for attributes
- **Supports types of Elements Global / Local Element types:** Scope of the element declarations
- XML Schema Namespace : <http://www.w3.org/2001/XMLSchema>



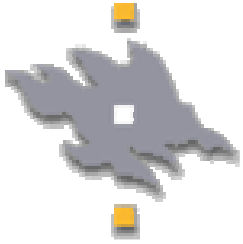
What is XML Schema ?

- XML Schemas purpose is to define a class of XML documents and specifically the structure of such documents and the data types used within the XML documents.
 - The XML Schema addresses both of them via two specifications Parts XML Schema-Part-I Structures and XML Schema Part-II Datatypes
- The documents which conform to the definition of the XML schema are called as Instance Documents



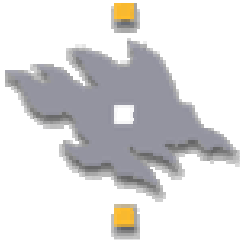
XML Schema Document

- This is an XML document which defines the schema for a class of XML documents, Similar to DTDs but richer from the structural and data types aspects.
- The root element of a Schema document is "schema" belonging the XML Schema Namespace :
<http://www.w3.org/2001/XMLSchema>
- XML Schema defines two types :
 - **SimpleType:** An XML elements which can not have any subelements or even attributes. They contain simple data types, enumeration, lists, restricted formatted values(patterns/regular expression evaluated)
 - **Complex Types:** Allow elements with or without subelements to be defined within them and they can have attributes.



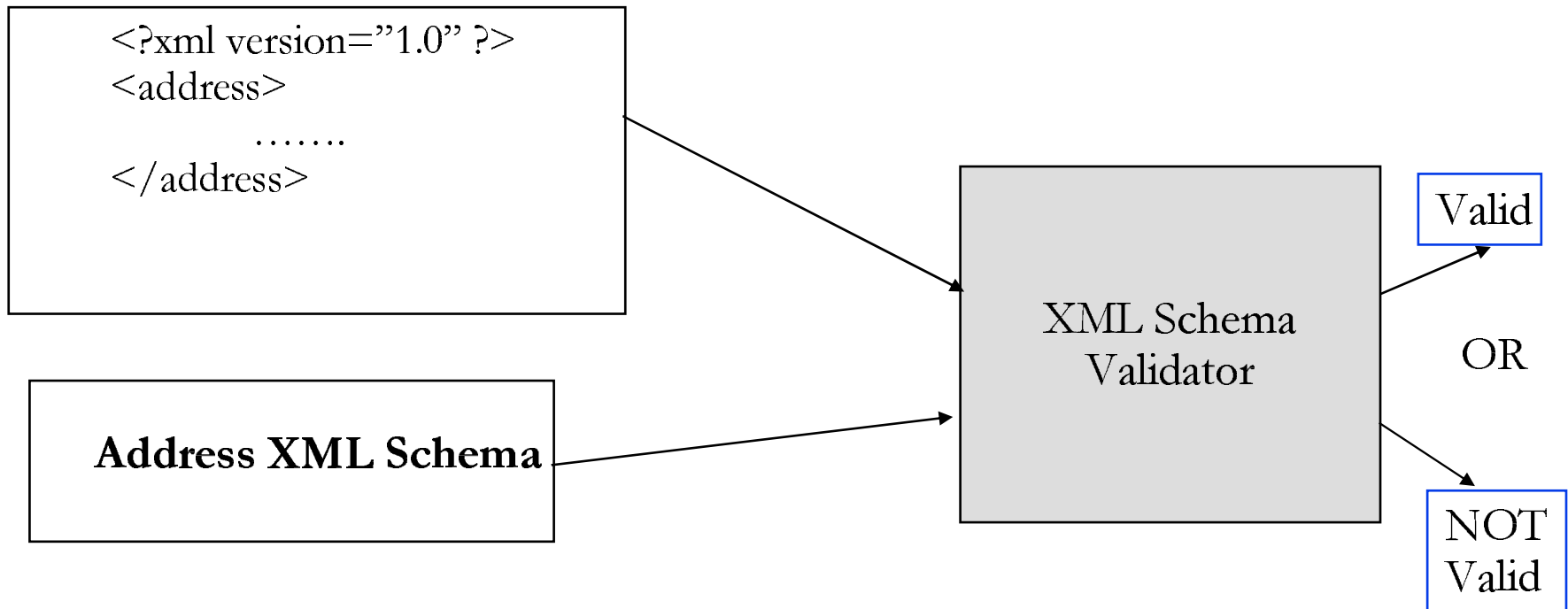
XML Schema benefits

- They are defined as XML Documents by Themselves
- They enable reuse or extensions to already defined Schemas a bit like object oriented languages
- They have richer document structure definition capabilities
- XML Schemas provide a rich set of data definition capabilities: ~44 different types, data ranging, data formatting, pattern definition(regular expressions, etc)
- The XML Parsers do not require to use a different parsing techniques inorder to validate XML documents
- They are Namespace aware hence leveraging and reusing already well defined schemas.
- Allows creation of own data types

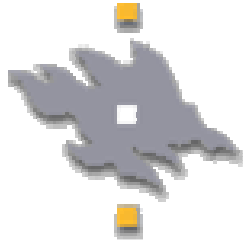


XML Schema validation

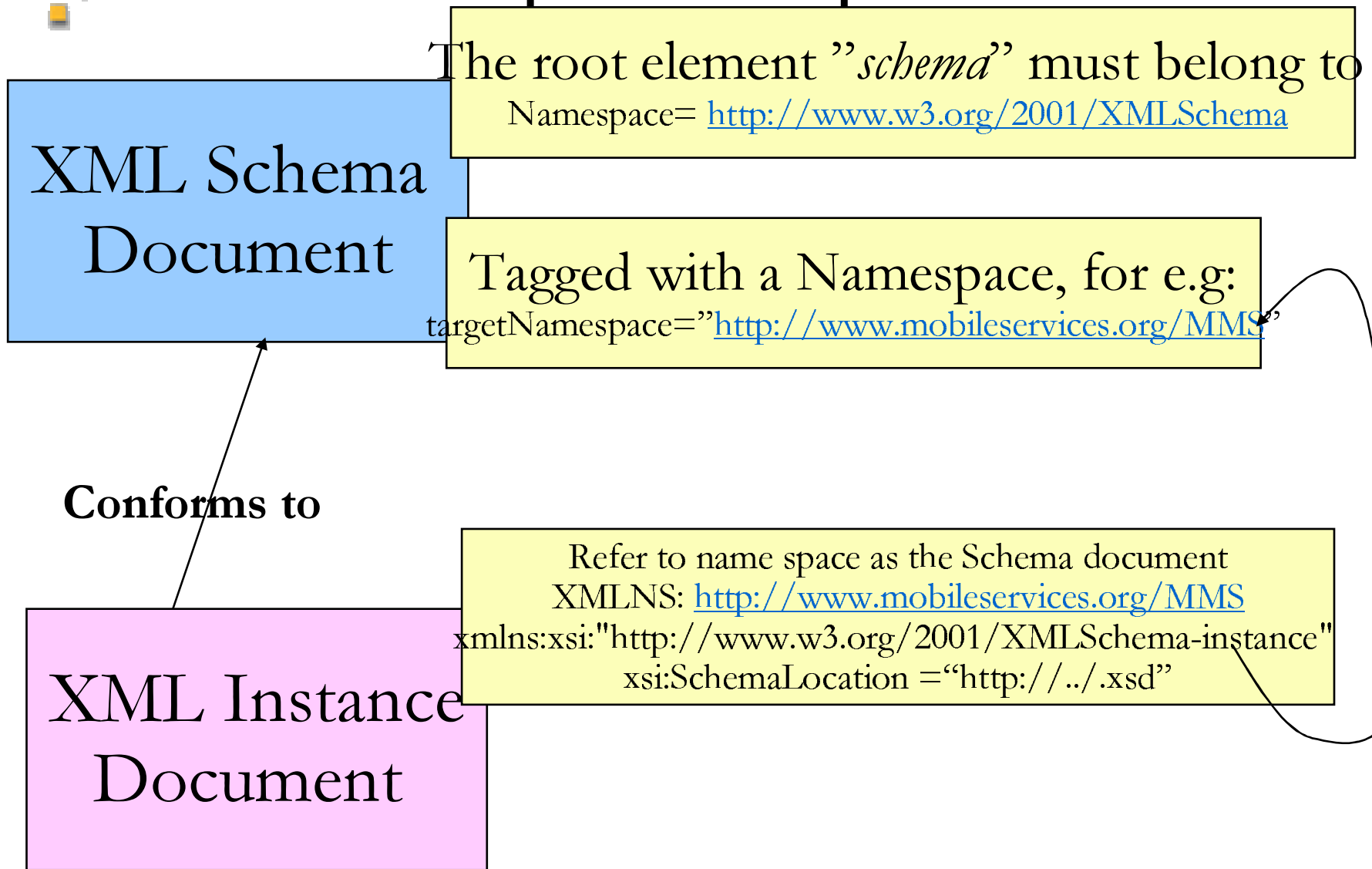
XMLInstance Document

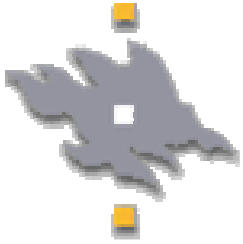


Note: It is not necessary for an XML instance document to explicitly reference an XML Schema document

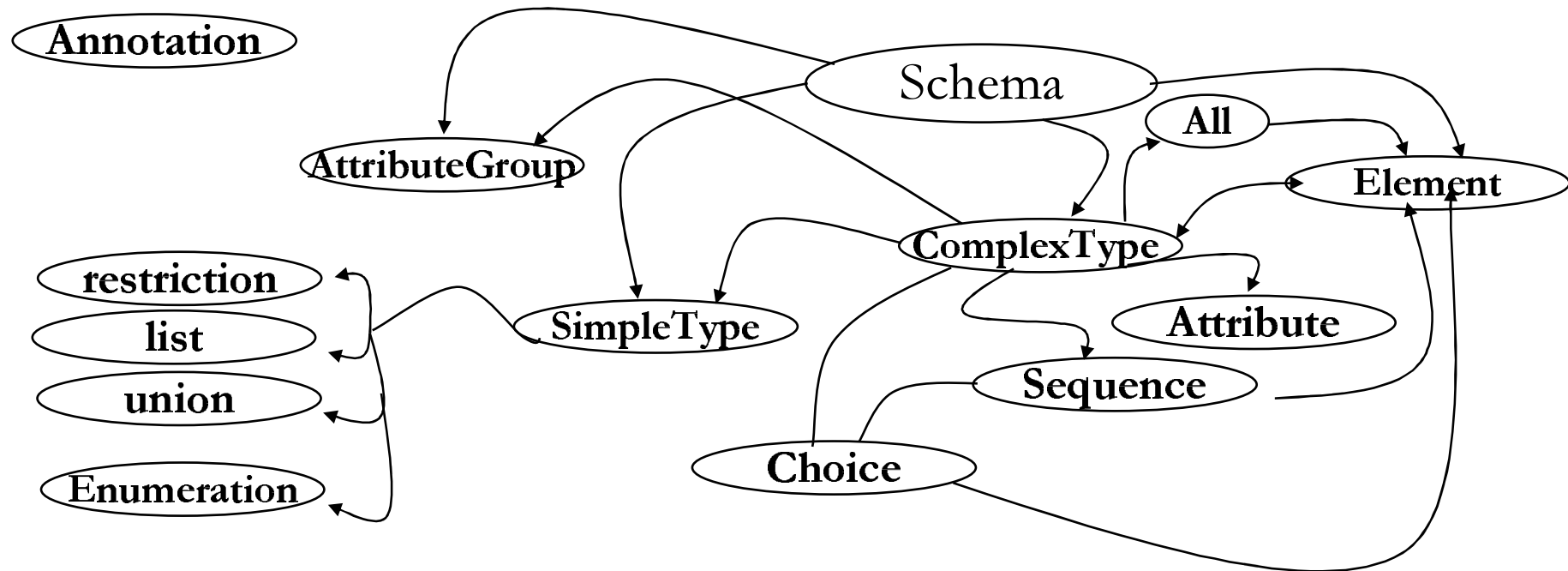


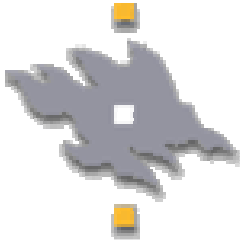
XML Schema and Instance





XML Schema Document

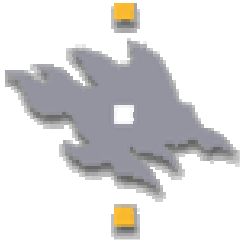




XML Schema Specification

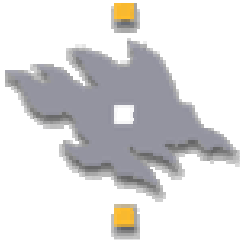
- XML Schema Structures
- XML Schema DataTypes





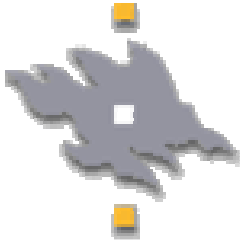
SOAP





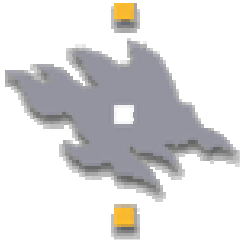
What is SOAP?

- SOAP was historically known as **S**imple **O**bject **A**ccess **P**rotocol, not anymore it is simply SOAP.
- SOAP is a simple and flexible messaging framework for transferring information specified in the form of an XML infoset between an initial SOAP sender and an ultimate SOAP receiver. SOAP **does not define** application semantics but **defines** a mechanism to express application messaging semantics.
- "SOAP defines a simple and lightweight mechanism for exchanging structured and typed information between peers over the web in a decentralized, distributed environment using XML."



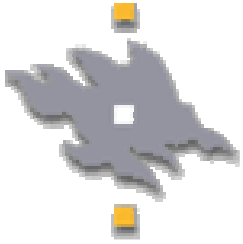
What is SOAP? [contd..]

- A lightweight protocol for exchange of information in a decentralized, distributed environment
- An XML based protocol that consists of three parts:
 - an envelope that defines a framework for describing what is in a message and how to process it,
 - a set of encoding rules for expressing instances of application-defined datatypes,
 - and a convention for representing remote procedure calls / document or message objects and corresponding responses.



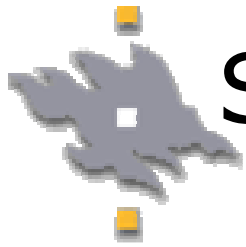
SOAP Terminology

- **Node:** Enforcing the rules that govern the exchange of SOAP messages. It accesses the services provided by the underlying protocols through one or more SOAP bindings.
- **Role:** A SOAP node's expected function in processing a message(next, none, ultimate Receiver)
- **Binding:** Formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange
- **Feature:** Extension of the SOAP messaging framework
- **Module:** Specification that contains the combined syntax and semantics of SOAP header blocks



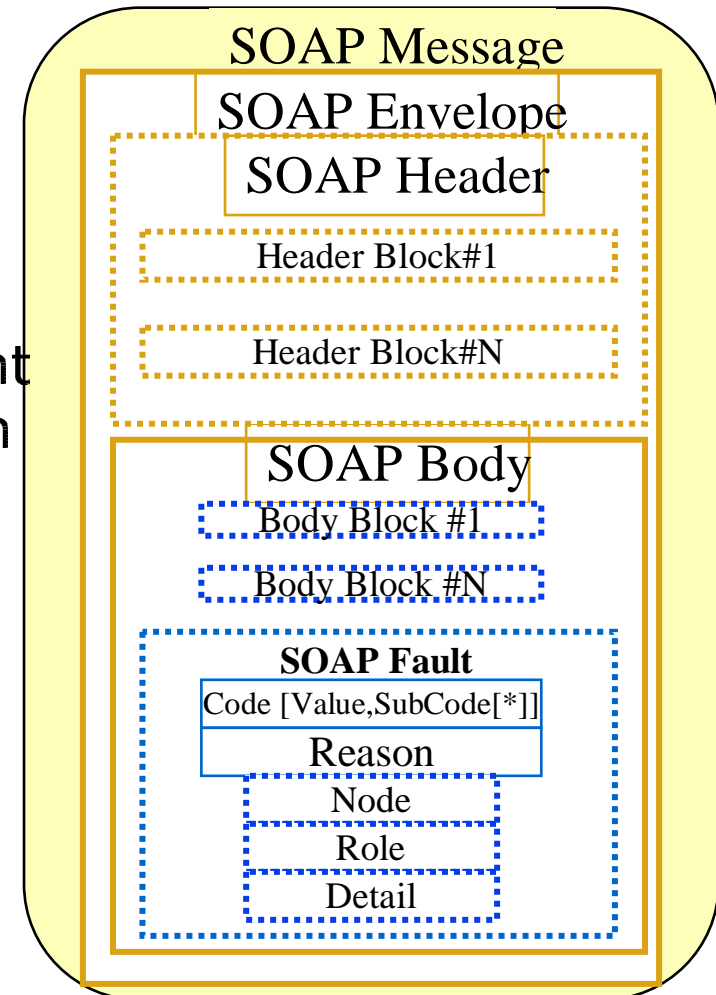
SOAP Terminology

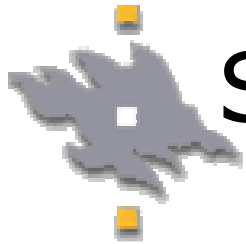
- **MEP:** Exchange of SOAP messages between SOAP nodes enabled by one or more underlying SOAP protocol bindings
- **SOAP Application:** software entity that produces, consumes or otherwise acts upon SOAP messages in a manner conforming to the SOAP processing model
- **Message Path:** Set of SOAP nodes through which a single SOAP message passes



SOAP Message Structure (1.2)

- **Envelope:** This is top level root element of a SOAP Message, which contains the Header and Body element.
- **Header:** A collection of zero or more SOAP header blocks each of which might be targeted at any SOAP receiver within the SOAP message path.
- **Body:** A collection of zero or more *element information items* targeted at an ultimate SOAP receiver in the SOAP message path

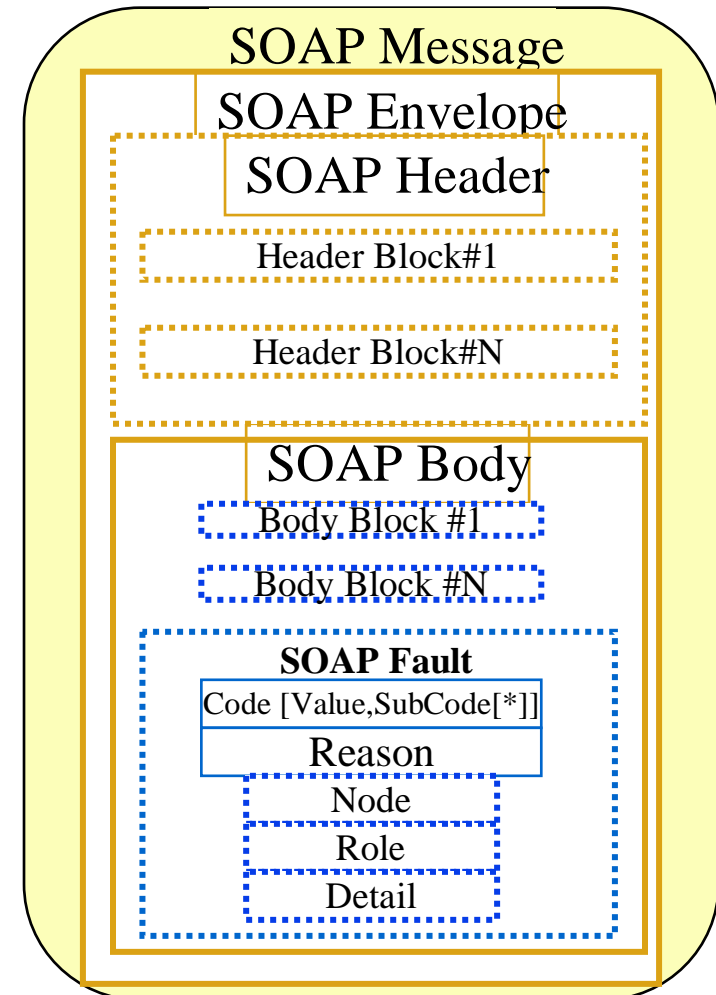


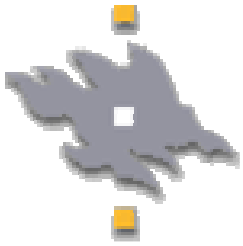


SOAP Message Structure (1.2)

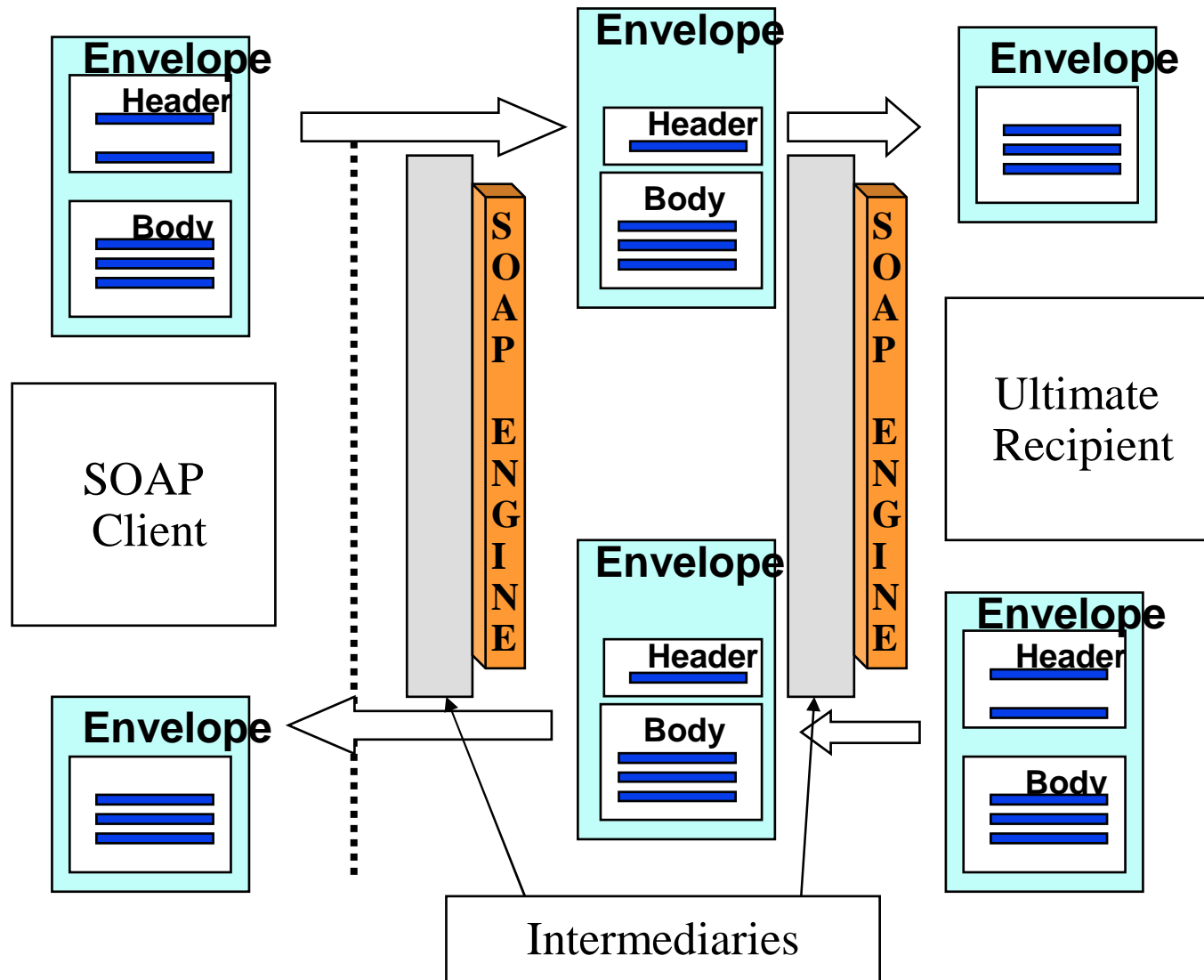
- **Fault:** A SOAP *element information item* which contains fault information generated by a SOAP node
 - **Code:** contains a highlevel code classifying the nature of the fault
 - **Reason:** Intended for human readable text
 - **Node:** SOAP node on the SOAP message path caused the fault
 - **Role:** The role the node was operating in at the point the fault
 - **Detail:** intended for carrying application specific error information

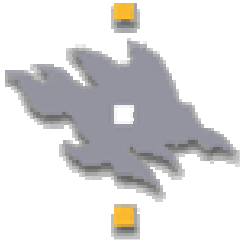
Note: All the above SOAP1.2 Message elements must be defined by the Namespace :
<http://www.w3.org/2003/05/soap-envelope>





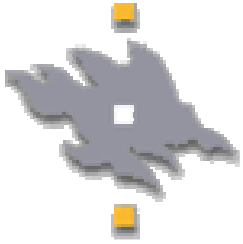
SOAP - Intermediaries





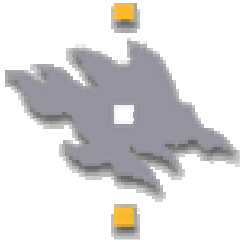
WSDL





Some background

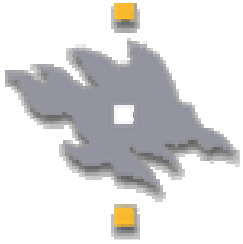
- Web Interface Definition Language ([WIDL](#)), by WebMethods, was one of the earliest ones, targeted as it was towards first-generation Web services systems such as XML-RPC and WDDX[w3C Note 22 Sept 1997]
- WIDL mimicked Interface Definition Languages that are the basis of CORBA and COM, but in XML form



Some background

Microsoft :

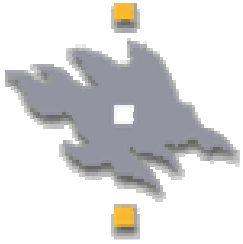
- Microsoft developed Service Description Language (SDL) and SOAP Contract Language (SCL) in 2000
- Developed a system for syndicating information about Web services: Discovery of Web Services (DISCO),



Some background

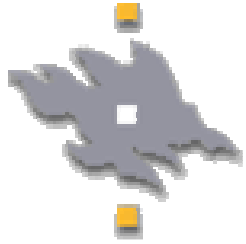
IBM :

- Network Accessible Service Specification Language (NASSL) and Advertisement and Discovery of Services (ADS), which paralleled SDL and SCL,



Some background

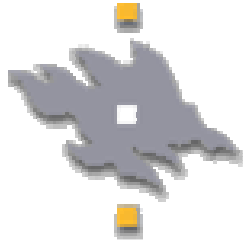
- Similar efforts from Ariba and other companies (IBM, Microsoft & Ariba), led into the Web Services Description Language -> [WSDL](#)
- Same group started to merge their various proprietary discovery systems into Universal Description, Discovery and Integration ([UDDI](#))



Web Services Description Language (WSDL)

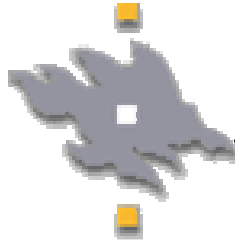
Originally submitted to W3C on 15 Mar 2001 by the Ariba, Microsoft and IBM and WSDL1.2 is currently a Working draft at W3C

- WSDL provides an XML based grammar to define web services as a collection of the communication end points capable of exchanging messages (procedural or document oriented) which follow a specified message structures in order to satisfy the functioning of a software system over a networked environment.



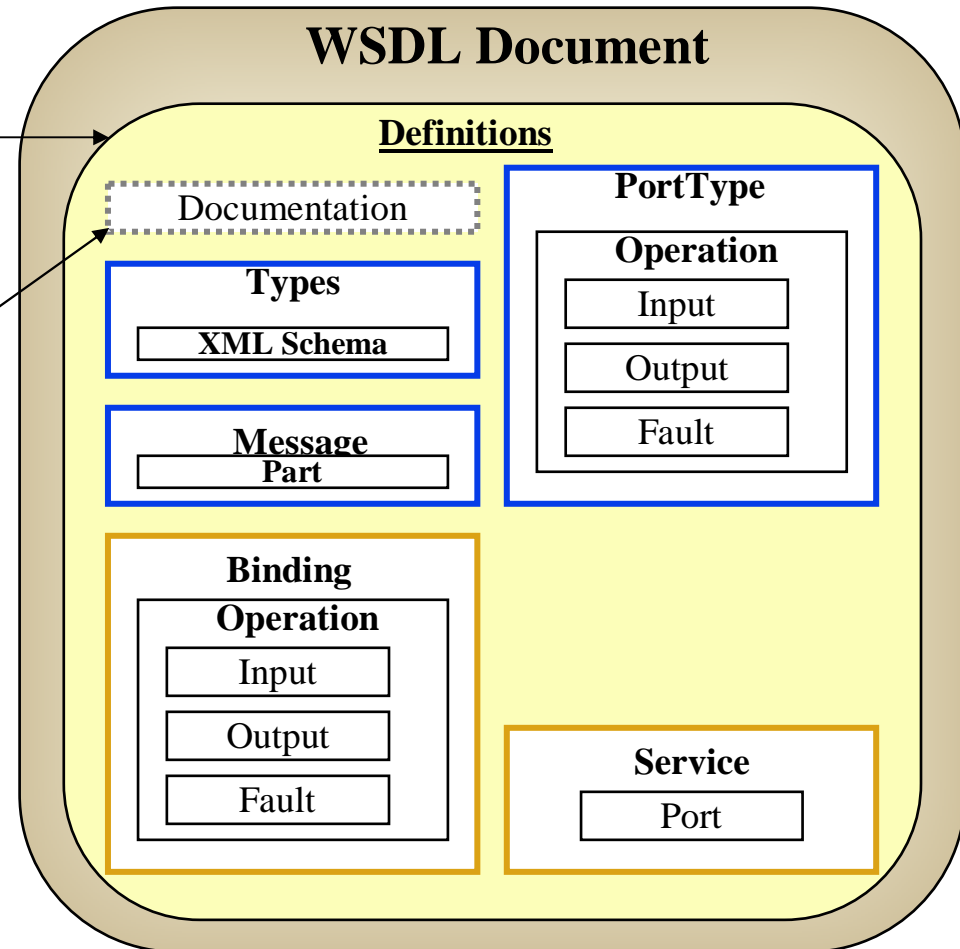
Web Services Description Language (WSDL)

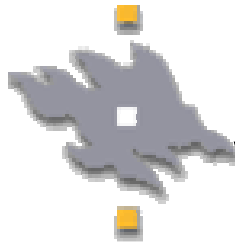
- The description is modeled into
 - Abstract : containing end points and messages
 - Concrete parts : binds these endpoints and messages to a concrete network deployment and data format bindings
- In principle this split in the service description promotes **multiple deployment models of the baseline service definitions.**



WSDL1.1 Document Structure

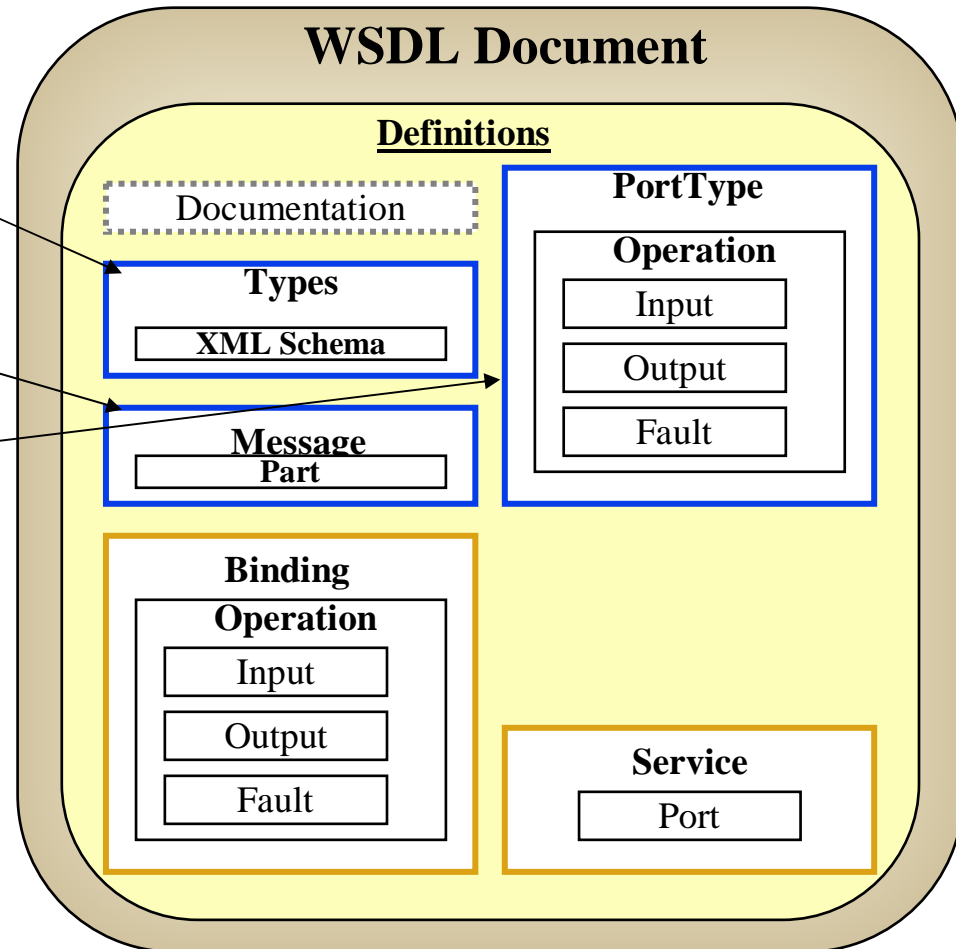
- "*Definitions*" is the root element which contains the complete service definition
- "*Documentation*" is an Optional sub element used for documentation which is human readable

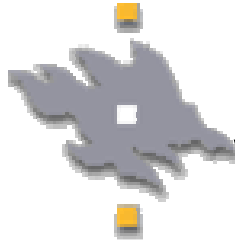




WSDL1.1 Document Structure

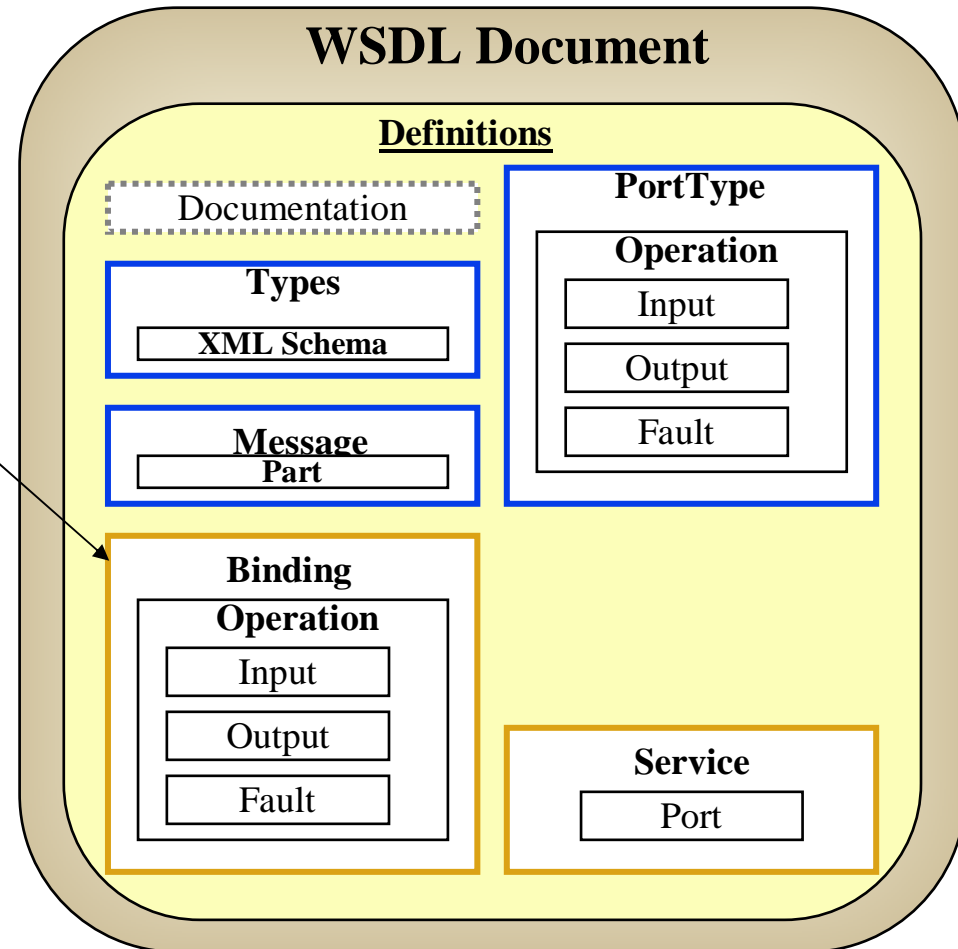
- **Types:** Data type definitions (XML Schema data types)
- **Messages:** Abstract definition of the data being transmitted
- **PortType:** Set of Abstract Operations,
 - Operation : each operation refers to Input, Output and Faults
 - Input
 - Output
 - Fault

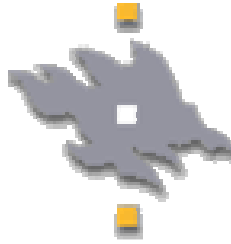




WSDL1.1 Document Structure

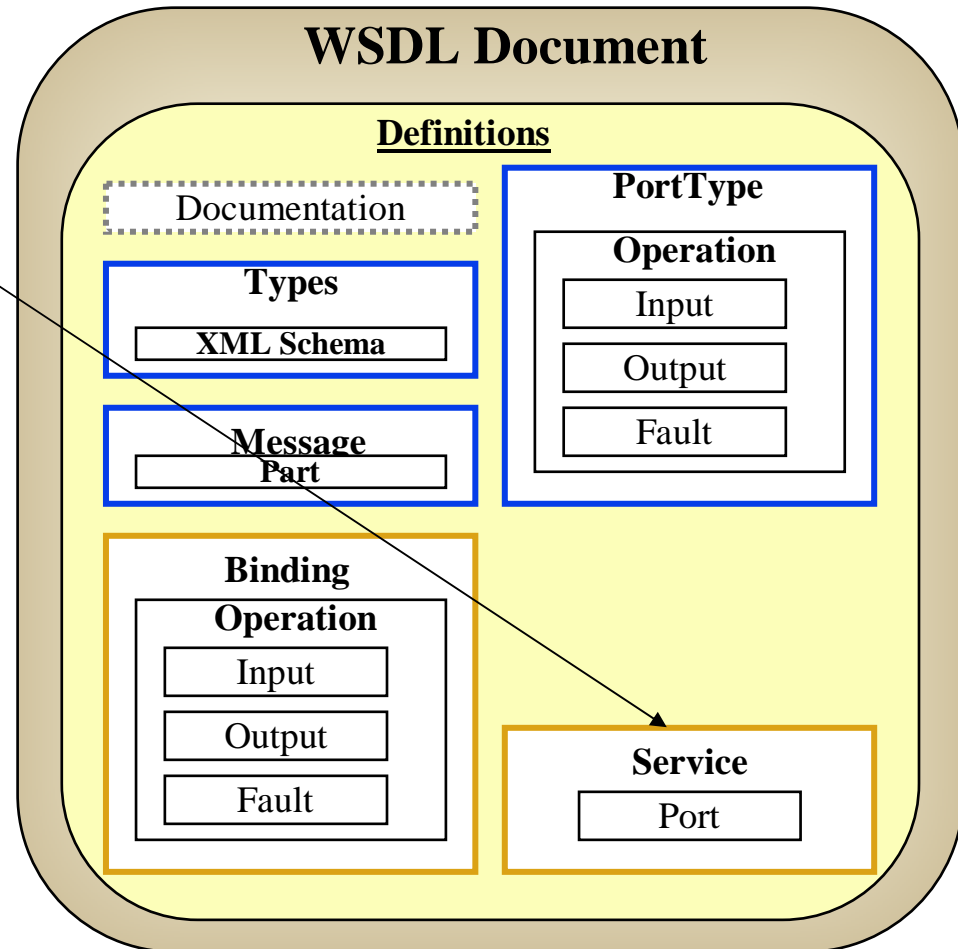
- **Binding:** Specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.
 - Operation
 - Input
 - Output
 - Fault

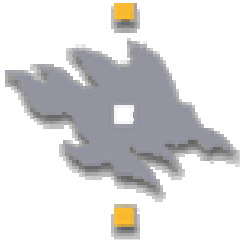




WSDL1.1 Document Structure

- **Service:** Utilized to aggregate a set of related ports
 - **Port:** Specifies an address for a binding

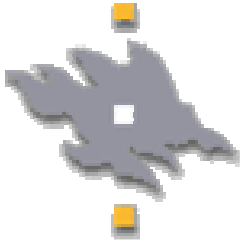




WSDL Bindings

WSDL Provides 3 different bindings currently,
namely:

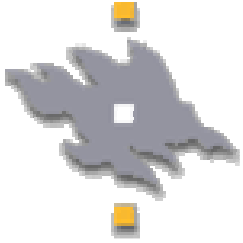
- SOAP
- HTTP 1.1 GET and POST Bindings
- MIME Binding



WSDL Bindings

WSDL Provides 3 different bindings currently, namely:

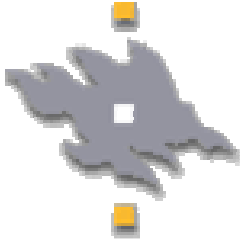
- SOAP
 - An indication that a binding is based on SOAP 1.1(/2) protocol.
 - Address specification for SOAP endpoints.
 - A SOAP Action URI basically ending up as an HTTP header for the HTTP binding of SOAP.
 - Header, Body and Fault definitions



WSDL Bindings

WSDL Provides 3 different bindings currently, namely:

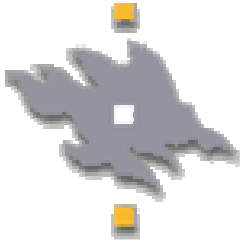
- HTTP 1.1 GET and POST Bindings
 - Non -SOAP based binding
 - Specify the base URI as a port (<http://www.mobileservices.org>) and the specific functionality to be invoked executed in the operations (`/services/delivery/deliver`)



WSDL Bindings

WSDL Provides 3 different bindings currently, namely:

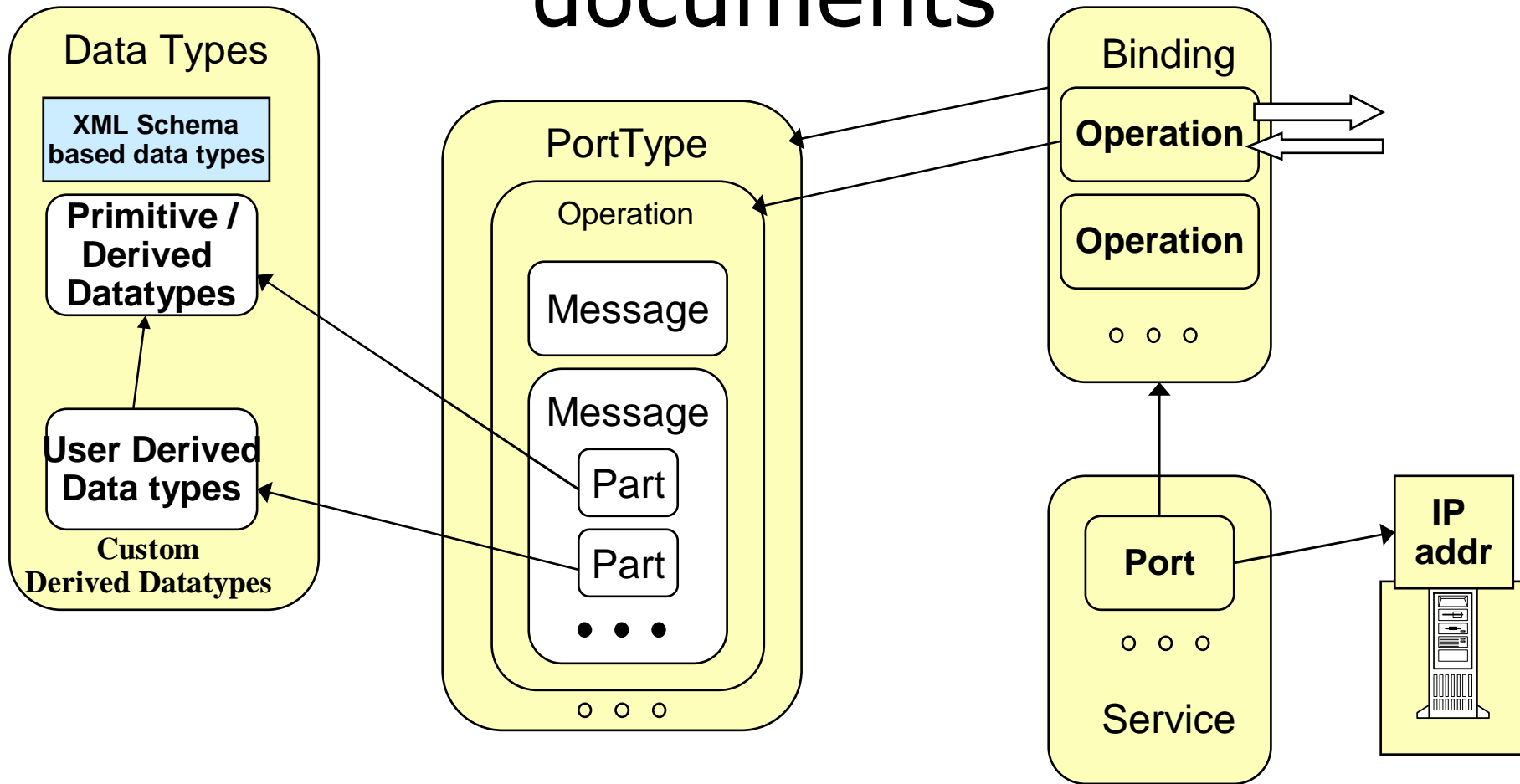
- MIME Binding
 - Allows the specification of multipart content being either part of input/output/fault.
 - Relies on either SOAP or HTTP binding as defined above.



WSDL namespaces used

- WSDL Namespace: <http://schemas.xmlsoap.org/wsdl/>
- Binding namespaces:
 - SOAP1.1 binding namespace:
<http://schemas.xmlsoap.org/wsdl/soap/>
 - HTTP binding namespace: <http://schemas.xmlsoap.org/wsdl/http/>
 - MIME Binding:
<http://schemas.xmlsoap.org/wsdl/mime/>

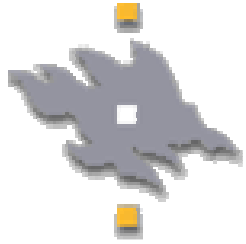
Modularization of WSDL documents



Datatyping

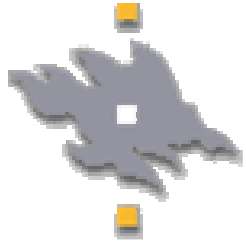
Abstract Interface

Concrete Interface



WSDL definition Reusability

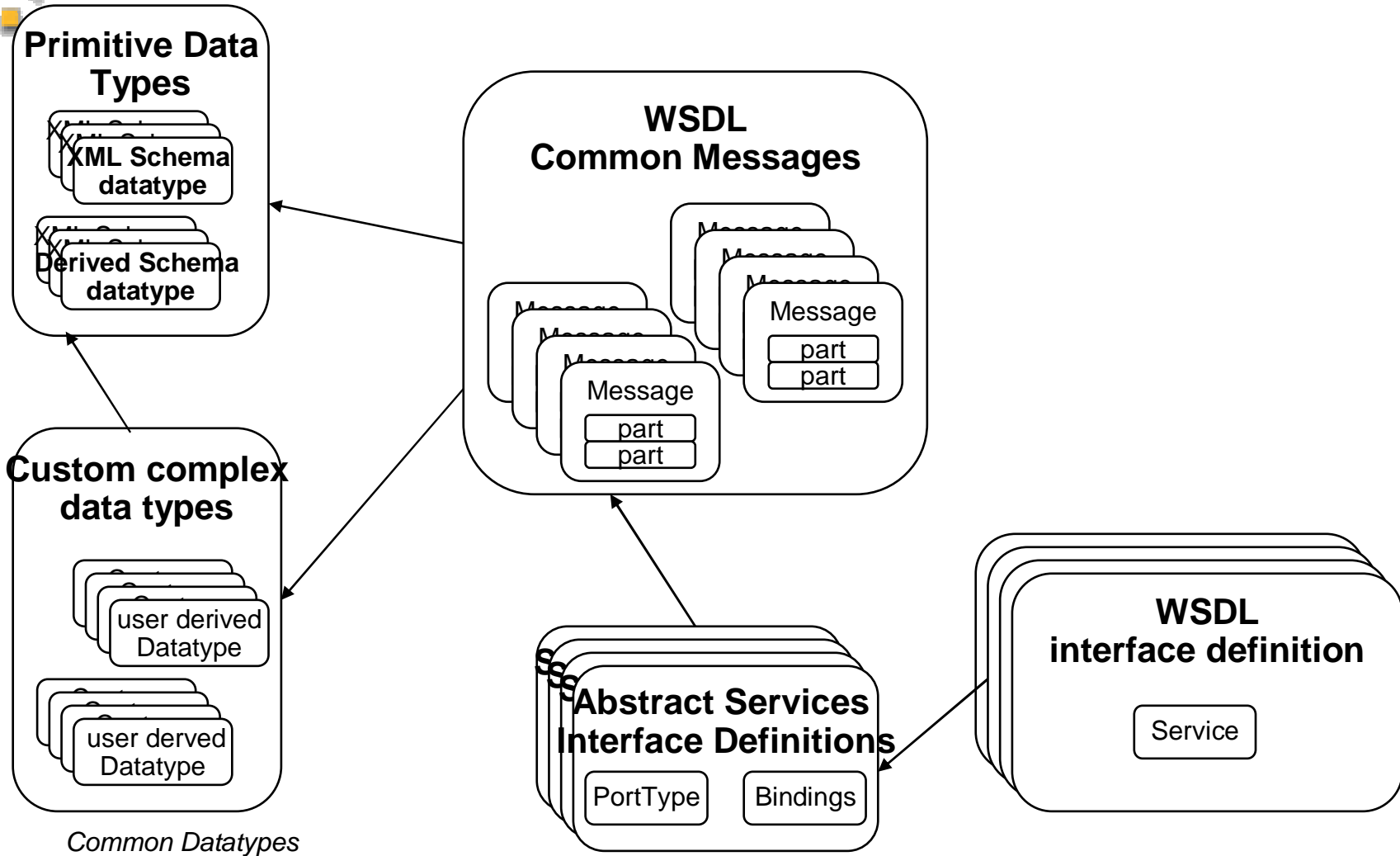
- WSDL through its "import" element construct enables reusability
 - Helps writing clean and manageable service descriptions in a modular fashion separating the definitions according to their level of abstraction.
 - It increases the ability to reuse definitions from a common set of service definitions.

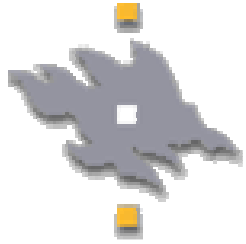


WSDL definition Reusability

- WSDL through its "import" element construct enables reusability
 - This is particularly beneficial when considering a family of WSI specification which are related to a particular industry domain.
 - Enables definition of reusable Web Services features which address issues such as: Security, Routing, Reliability, Authorization & Authentication, Addressing, Eventing, and more..

WSDL Reusability – An Example

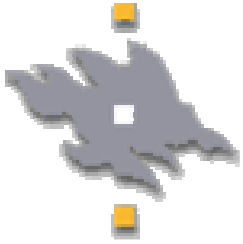




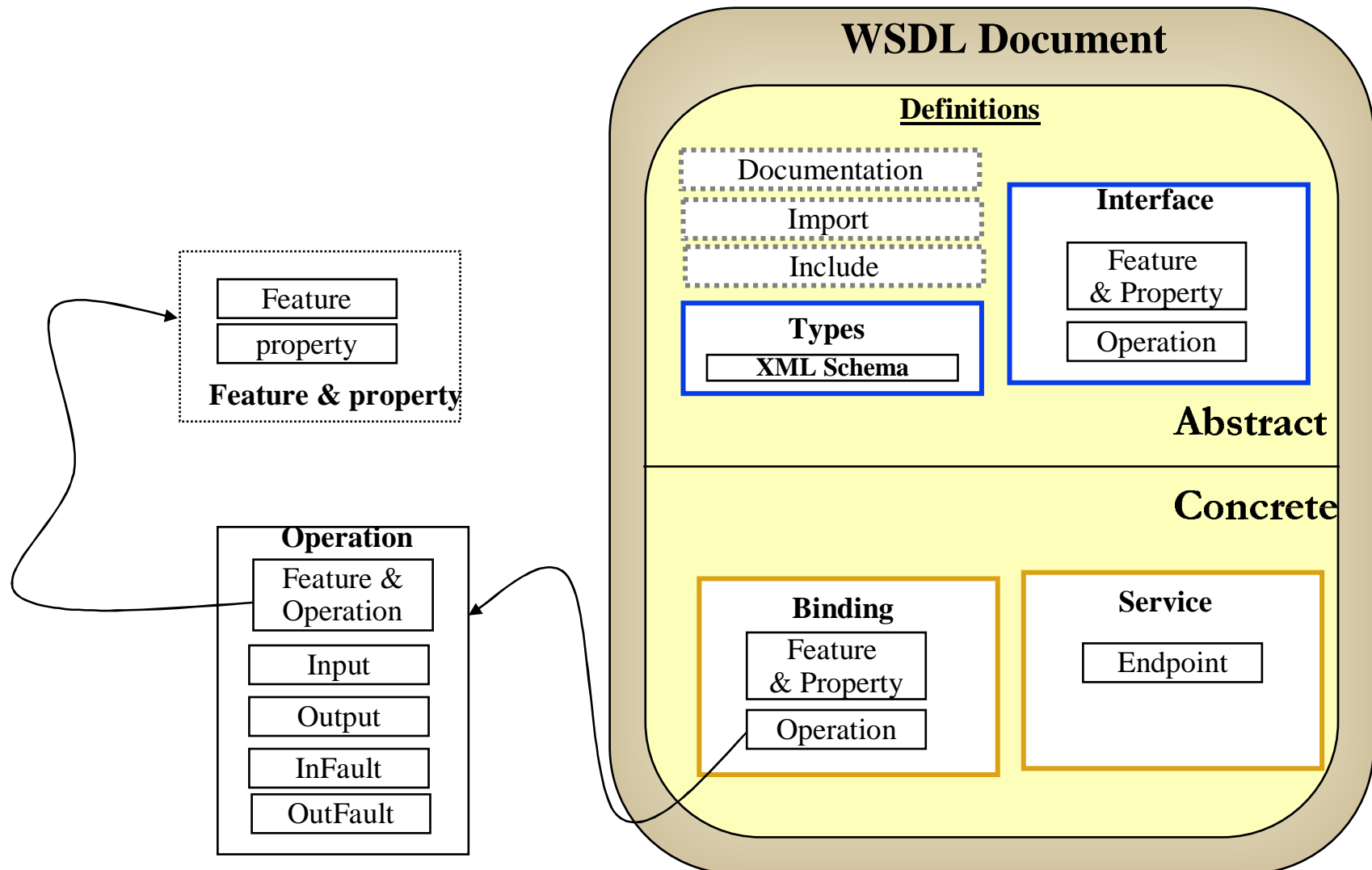
What's new with WSDL 2.0?

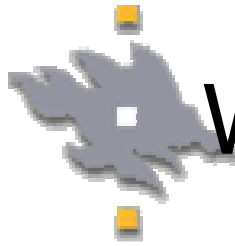
- New Namespaces
- portType changed to Interface
- **Operation has new elements:** Feature, Property, Infault and outFault
- Interfaces can extend from another interface, basically inheriting all previously defined operations and features set
- Message referencing
- Separation of Faults: inFault and outFault

- **Operation Styles:** RPC, Get-Attribute, Set-Attribute
- **Multiple Message Exchange Patterns:** In-Only, Robust In-Only, In-Out, In-Multi-Out?, Out-Only, Robust-OutOnly, Out-In, Asynchronous Out-In, Out-Multi-In?

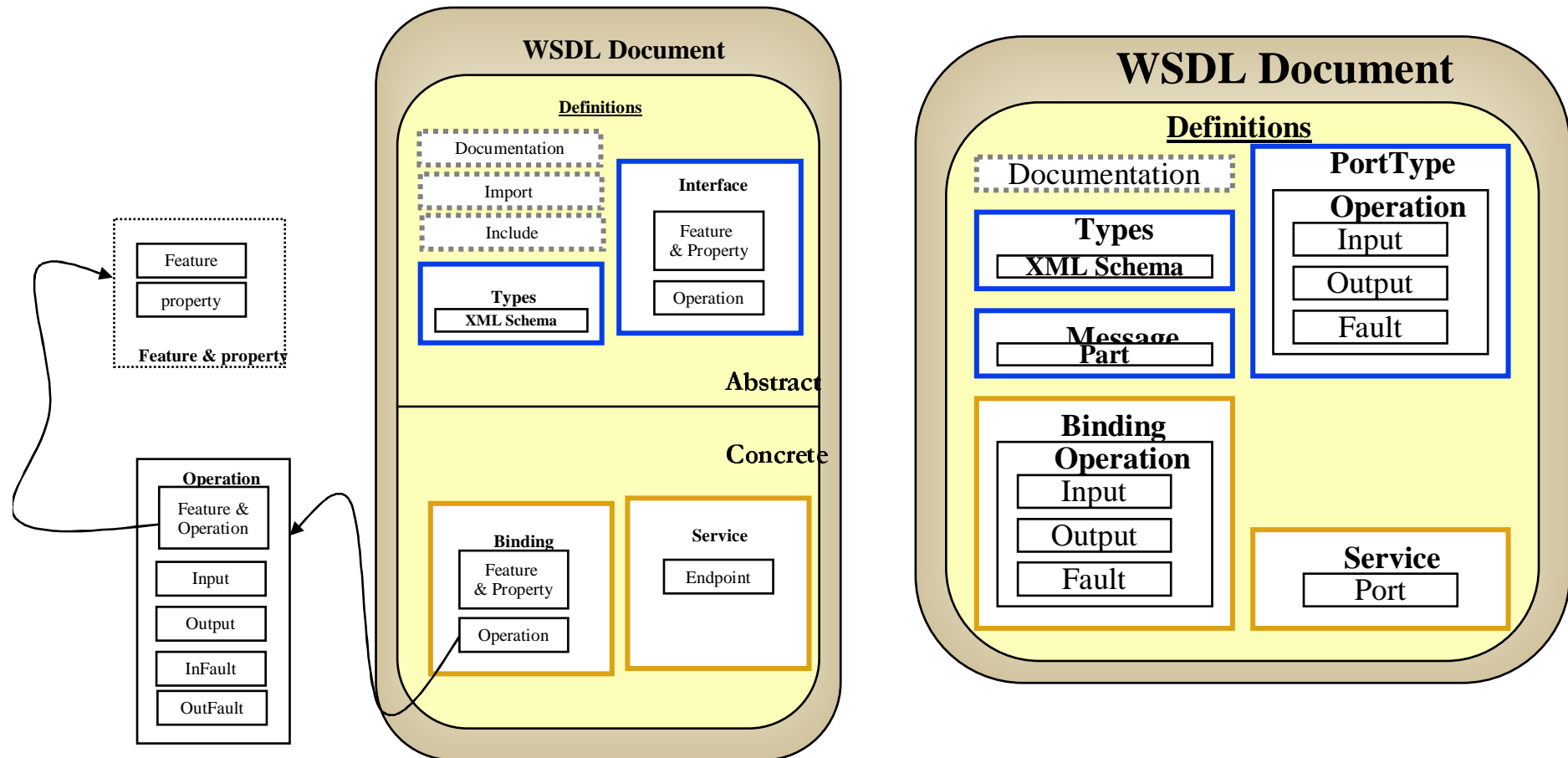


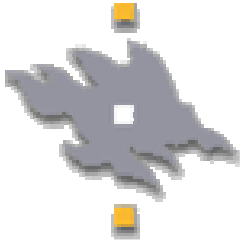
WSDL Document Structure





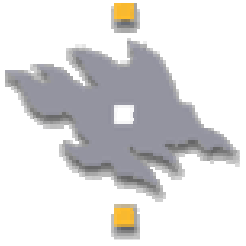
WSDL2 document Structures in comparison to WSDL1.1





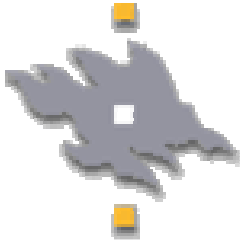
WSDL namespaces used

- WSDL Namespace: <http://www.w3.org/2003/11/wSDL>
- Binding namespaces:
 - SOAP1.2 binding namespace:
<http://www.w3.org/2003/11/wSDL/soap12>
 - HTTP binding namespace:
<http://www.w3.org/2003/11/wSDL/http>
 - MIME Binding:
<http://www.w3.org/2003/11/wSDL/mime>



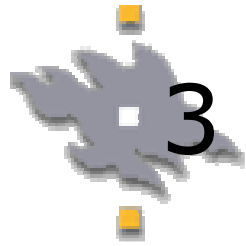
UDDI





UDDI

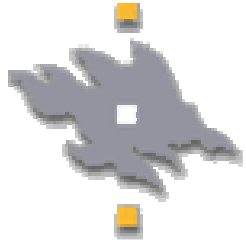
- UDDI Stands for **U**niversal **D**escription, **D**iscovery & **I**ntegration: It enables businesses to quickly, easily, and dynamically find and transact with one another
- Web Services are being deployed today and UDDI today provides the best way to publish and discover these webservices, by providing a business registry.
- UDDI provides means to:
 - A). **Describe** a business and its services
 - B). **Discover** other businesses offering desired services
 - C). **Integrate** with these other businesses.



3 Components of UDDI business registry

- **White Pages:** Company Contact information(name, contacts, human readable description and identifiers)
- **Yellow Pages:** Categorization of businesses using standard taxonomies (Industry codes, Geographic indexes)
- **Green Pages:** technical information about the services that are published(Service description, business rules, application invocations, data binding).

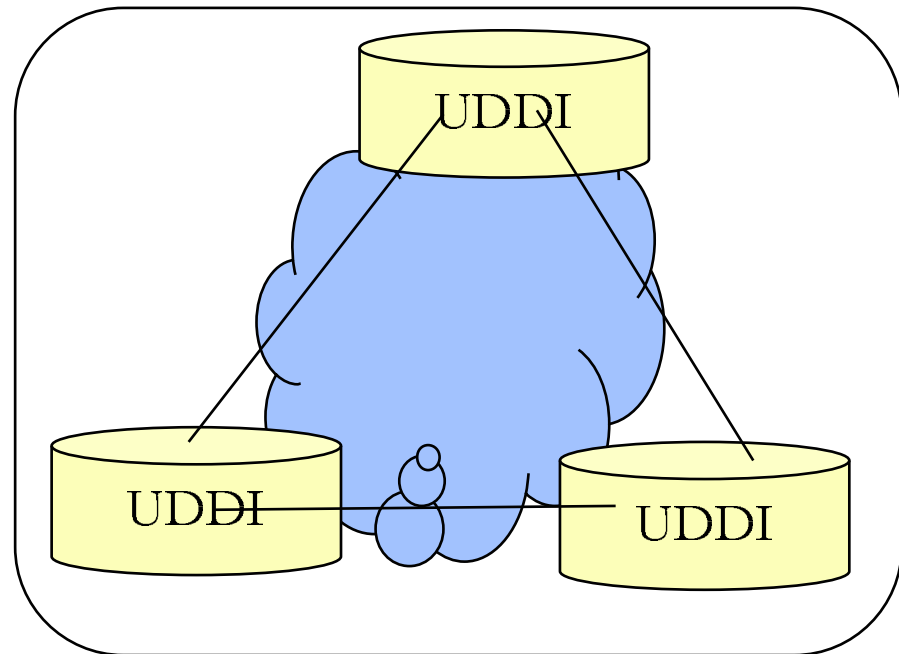


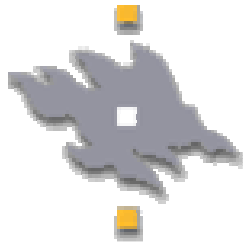


Logical Centralized Business Registry

UDDI registries are physically distributed, but replicate data amongst a set of partnered registries on a regular basis

Publishing in any one registry will be automatically replicate across multiple registries , hence providing a view of **Logically a single business registry**

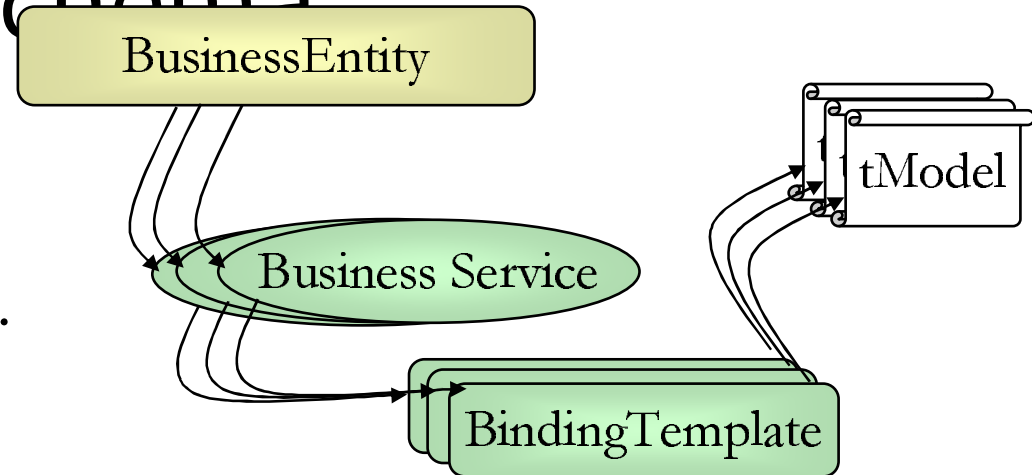




Four Basic Elements of UDDI Schema

UDDI utilizes XML Schema to describe information about services. The namespace is depicted by URN: **urn:uddi-org:api_v2**

The Schema defining a specific Business services can be found here
http://uddi.org/schema/uddi_v2.xsd



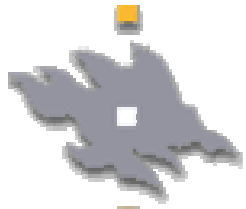
The Schema consists of four major components:

BusinessEntity

BusinessServices

BindingTempate

tModel

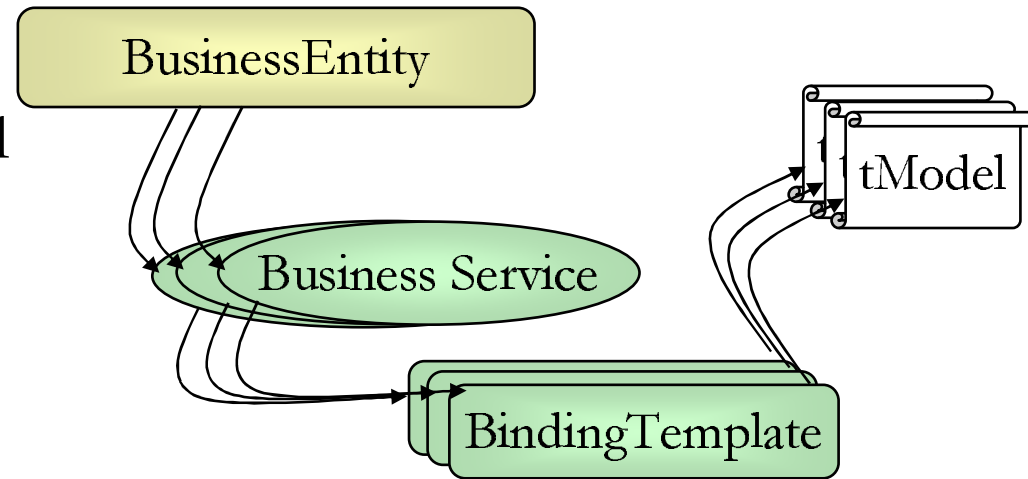


Four Basic Elements of UDDI Schema

BusinessEntity: Defines the yellow pages containing business name, identifiers, categorization, contacts. This forms the top level information for all details related to a particular business

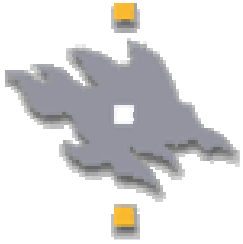
BusinessServices: A container in order to group a set of web services and categorized according to product, industry, service and geographical locations

BindingTemplate: Contains information to invoke a service. This contains references to



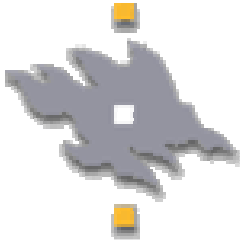
specifications the service complies to.

tModels: Metadata about specifications, which includes the name, publisher details and URL to the actual specification



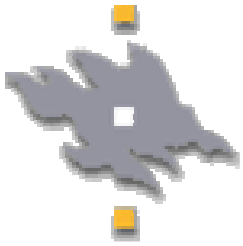
UDDI's provided APIs

- UDDI provides a SOAP based API to the business registry.
- UDDI provides two types of APIs, namely:
 - **Inquiry APIs** : Browse, Drilldown & Invocation patterns
 - Find & Get details of : binding, business, relatedBusiness, service, toModel
 - **Publish APIs** : Add, Get , Delete & Save(BusinessEntity, businessService, bindingTemplate and tModel), : publisherAssertions, Authtokens,

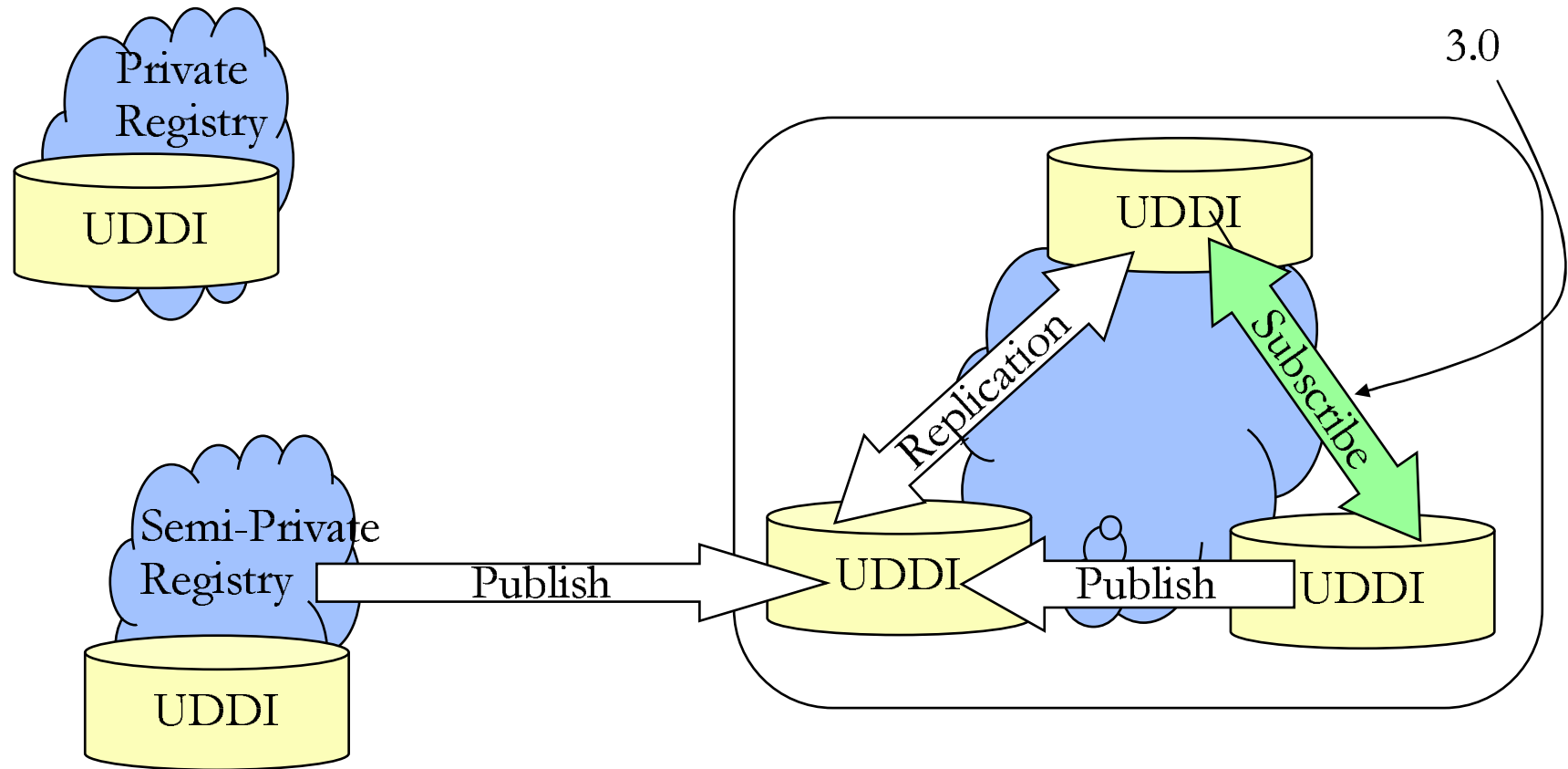


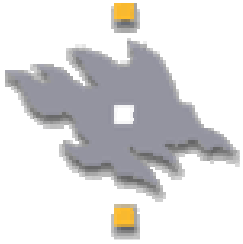
UDDI Security

- Publisher API are accessible to authenticated users
- Inquiry API are accessible to any user
- Changes to published information is only allowed to the creator of the original information.
- Each UDDI registry is allowed to create its own end user authentication mechanism



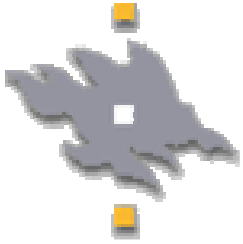
UDDI Deployment Models





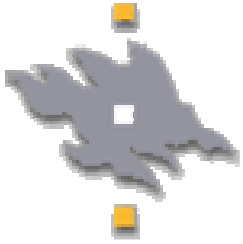
UDDI Evolution

- Microsoft Ariba and IBM released version 1.0 September 2000
- June 2001 UDDI version 2.0 released
- July 2002 UDDI version 3.0 released
- Universal Description, Discovery & Integration originally developed by 3 industry players, IBM, Microsoft & Ariba was submitted to OASIS in June 2002 along with UDDI version 2.0 and 3.0 for standardization



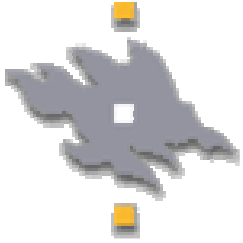
WSDL mapping to UDDI

- Considering the fact that the WSDL consists of two basic parts
 - The abstract reusable service interface definition
 - A Concrete binding of the abstract to a particular instance of the service interface
- In context with the UDDI only the abstract part of the WSDL is of immediate interest



Tmodel Example

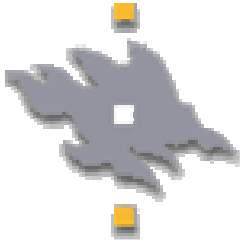
```
<tModel authorizedName="..." operator="..." tModelKey="...">
  <name>Notification Service</name>
  <description xml:lang="en">
    This is the WSDL description for a Delivery Web Service Interface
  </description>
  <overviewDoc>
    <description xml:lang="en">
      WSDL source document if needed
    </description>
    <overviewURL>
      http://www.mobile.services.org/services/notification.wsdl
    </overviewURL>
  </overviewDoc>
  <categoryBag>
    <keyedReference tModelKey=" uuid:q9Kfg46S-7639-7834-9838-48S2479Q93NE6 "
      keyName="uddi-org:types"
      keyValue="wsdlSpec"/>
  </categoryBag>
</tModel>
```



Business Service

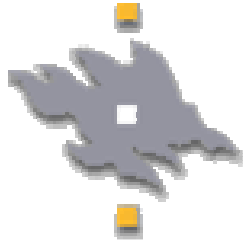
```
<businessService businessKey="..." serviceKey="...">
  <name>NotificationService</name>
  <description> (...) </description>
  <bindingTemplates>
    <bindingTemplate>
      (...)
      <accessPoint urlType="http">
        http://www.mobile.services.org/services/notification
      </accessPoint>
      <tModelInstanceDetails>
        <tModelInstanceInfo tModelKey="...">

          </tModelInstanceInfo>
        <tModelInstanceDetails>
      </bindingTemplate>
    </bindingTemplates>
  </businessService>
```



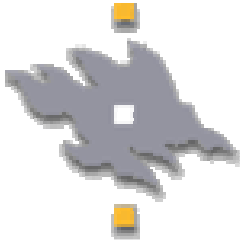
Web Services Architectures

- Standards defined around the *Web Services* technologies provide the required level of convention to maintain the required inter-operability (WS-I) between the Web Services.
- These current standards are not sufficient for the Service oriented enterprise systems to emerge and evolve based on them as they only help in developing inter-operable solutions, but do not provide the necessary infrastructure requirements.
- The Web Services standards are numerous and are also to some extent overlapping due to competing proposals from IT Vendors encouraged by potential IPRs behind the proposals



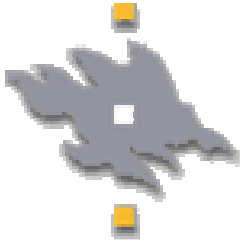
Web Services Architectures

- There is a need for a Web services infrastructure which can guarantee the basic needs for Web Services oriented eco systems, the basic needs being :
 - Security,
 - Authorization,
 - Debugging capabilities
 - Non-repudiation
 - Real time availability and reliability
 - Service level agreements,
 - Failure recovery,
 - Monitoring and logging
 - Billing & Metering



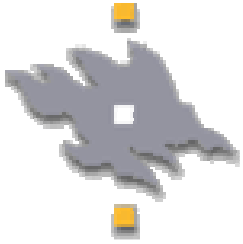
Web Services Infrastructure

- This is a run time environment over which the Service providers, brokers and consumers can inter-operate with trust ensuring that the different players are operating in a predictable and reliable fashion
- Web Services infrastructure could be hosted by corporate networks, network infrastructure providers, service integrators or the service providers.
- The players within the infrastructure comply with a set of rules, standards and build an element of trust within the networked environment



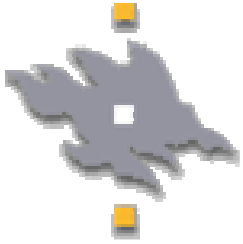
Web Services Infrastructure

- The players could be held account for malicious usage of the network or demand for the rightful services of such an infrastructure
- The basic services required within such an infrastructure are :
 - Security
 - Authentication and Authorization
 - Confidentiality
 - Connectivity
 - Management
 - Billing & Metering
 - Reliability



Security [Contd]

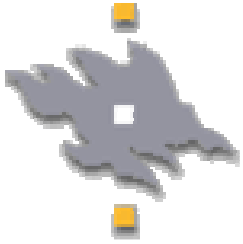
- **Authentication**
 - The infrastructure must provide a guaranteed identity of originator of the request and response messages
 - The originator is truly who they claim to be
- **Authorization**
 - Authorization to the rights and usage of a specific services within the infrastructure based policies and access control
- **Confidentiality**
 - The messages are untampered and are as originated or as intended to be or as they are expected to be as they are processed over multiple intermediaries and finally at the target recipient.
 - Information exchanged within the network environment is kept confidential and there is other use of the information exchanged between the participants of the Web Services within environment other than for the intended purpose agreed upon



Security

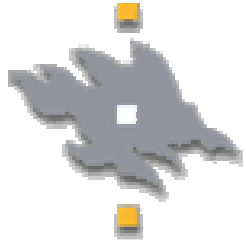
- Signatures
 - Digitally signed certificates are the best possible reliable means of authentication for both senders/receivers and message authentication

Note: Management of the authentication and the digital certificates and their seamless integration within a Web services framework is a key functionality that distinguishes Web services infrastructures



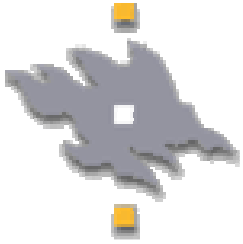
Connectivity & Reliability

- Guaranteed connectivity at all times over the Web Services Infrastructure for business critical application.
- The agreed reliability of the connection between the service participants plays a key role to success or failure.
- The WS infrastructure should be able to provide sufficient class of connectivity levels at all possible loads a business needs to perform efficiently.



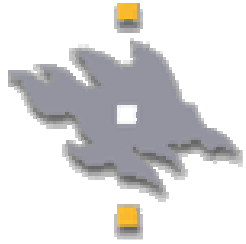
Web Services Management

- Provide simple and easy methods to manage Web Services within the WS Infrastructure:
 - Deploy
 - Monitor
 - Debug and
 - Maintain
- Simplify the Web Services by providing added services such as:
 - Logging and Tracing of Web Services utilization
 - Reporting the status of the Web Services
 - Providing default features within the infrastructure such as:
 - Billing
 - Charging



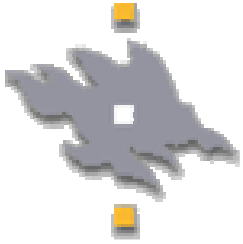
Web Services Reliability

- This infrastructure should ensure that the deployed Web Services are functioning in a reliable manner by ensuring that :
 - Guaranteed Delivery of messages to the intended recipient within the time critical factor.
 - Non-Repudiation, contain sufficient traces of the action so that recipient or the sender can not repudiate over the performed actions within the Web Services Environment.
 - Ensure the availability of functioning services at all times 24X7 as they are based on the Internet technologies, which are meant to be available at all times.
 - Maintain the level of confidentiality within the infrastructure environment.



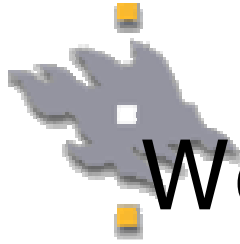
Web Services Architecture - Realization

- New Web Services technologies are being proposed very often as one identifies the lack of features to support in service oriented architectures in bring existing solutions into the Web Services World(Grids for example).
- All these technologies are providing relations to each other at a protocol level and in some cases overlapping with other specifications.
- These technologies require to be mapped to the context of the Web Services infrastructures at architectural level. There are considerable efforts been put into providing these technologies onto existing application platforms/development environment level(J2EE, .Net, etc), but there still lacks a clear Web Services Architecture for internal web services deployment & management



Web Services Development Models





Web Services Development Model

Top Down Approach

A. WSDL Creation:

- Identify & define the data types to be used (Custom as well as from the XML Schema DT base)
- Define the interfaces to be used
- Define the different operations & their bindings

B. Utilize tool to create stubs :

- Run through tool to create required automatically generated parts of the code

C. Implementation:

- Complete the implementation to receive the SOAP requests, process and deliver a Response incase required to do so.

D. Deploy the WS:

- Utilize the tools to deploy and host the WS
- Publish the WS into a registry like using UDDI, WSIL (optional)





Web Services Development Model

Bottom Up Approach (Implementation already exists)

A. Implementation

- If not already existing Develop a system using Java, C++ or any other programming language.

B. Create WSDL :

- Define the WSDL Manually and then map the SOAP requests to the API calls using the standard APIs or then
- Run through tool to create required automatically generated parts of the WSDL and the bindings (for example Java2WSDL, ApacheAxis).
- Integrate the generated parts

C. Deploy the WS

- Utilize the tools to deploy and host the WS
- Publish the WS into a registry like using UDDI, WSIL (optional)