

Prof. Dr.-Ing. Stefan Deßloch
AG Heterogene Informationssysteme
Geb. 36, Raum 329
Tel. 0631/205 3275
dessloch@informatik.uni-kl.de



Chapter 8 – Web Services Foundations



Types of E-Business

Business To Consumer (B2C)	Business To Business (B2B)	Intra Business
<ul style="list-style-type: none">• Relation between enterprise and customers• Sales-related aspects are predominant, like product presentation, advertising, service advisory, shopping	<ul style="list-style-type: none">• Relation between processes of different enterprises• Predominant are relation to suppliers, and customer relations to other enterprises like industrial consumers, retailers, banks	<ul style="list-style-type: none">• Electronic organization of internal business processes, like realization within workflow systems

B2B Integration – Conventional Middleware

- Middleware itself is (logically) centralized
 - usually controlled by a single company
 - now requires agreement on using, managing specific middleware platform across companies ("third party")
 - need to implement a "global workflow"
 - problems
 - lack of trust
 - autonomy needs to be preserved
 - business transactions are confidential
- Point-to-point solutions
 - lack of standardization
 - many partners involved -> heterogeneity of middleware platforms
- Focus on LAN
 - insufficient support for internet protocols
 - problems with firewalls
 - cannot work with multiple trust domains

Web Services

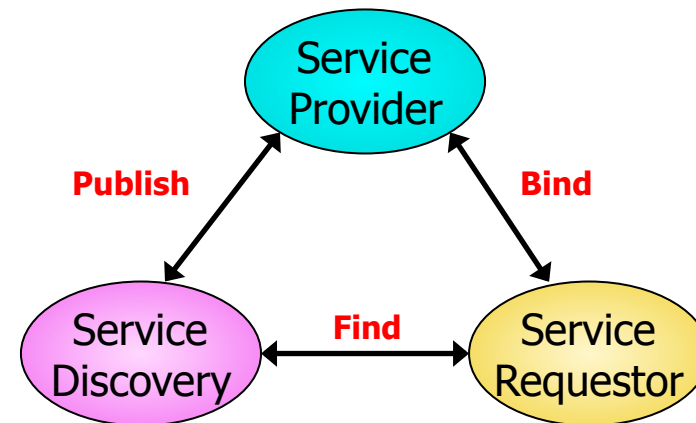
- New distributed computing platform built on existing infrastructure including XML & HTTP
 - Web services are for B2B what browsers are for B2C
- Self-contained, self describing, modular service that can be published, located and invoked across the web
 - Refer to open standards and specifications:
 - component model (WSDL)
 - inter-component communication (SOAP)
 - discovery (UDDI)
 - Platform- and implementation-independent access
 - Described, searched, and executed based on XML
- Enable component-oriented applications
 - Loose coupling from client to service
 - Enable to integrate legacy systems into the web
 - Useful for other distributed computing frameworks such as CORBA, DCOM, EJBs
 - ➔ **Web services as wrappers for existing IS-functionality**

Service-Oriented Architecture (SOA)

- **Service Requestor**
 - Finds required services via Service Discovery
 - Binds to services via Service Provider
- **Service Provider**
 - Provides e-business services
 - Publishes availability of these services
- **Service Discovery**
 - Service Registry
 - Provides support for publishing and locating services
 - Like telephone yellow pages
 - Service Index
 - Publication is "passive": service descriptions are made available and gathered by index service
 - Peer-to-peer Discovery
 - Dynamic discovery: requestor send queries to peers in a network

Definition (given by OASIS SOA Reference Model):

"A paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains"



Web Services - Definition

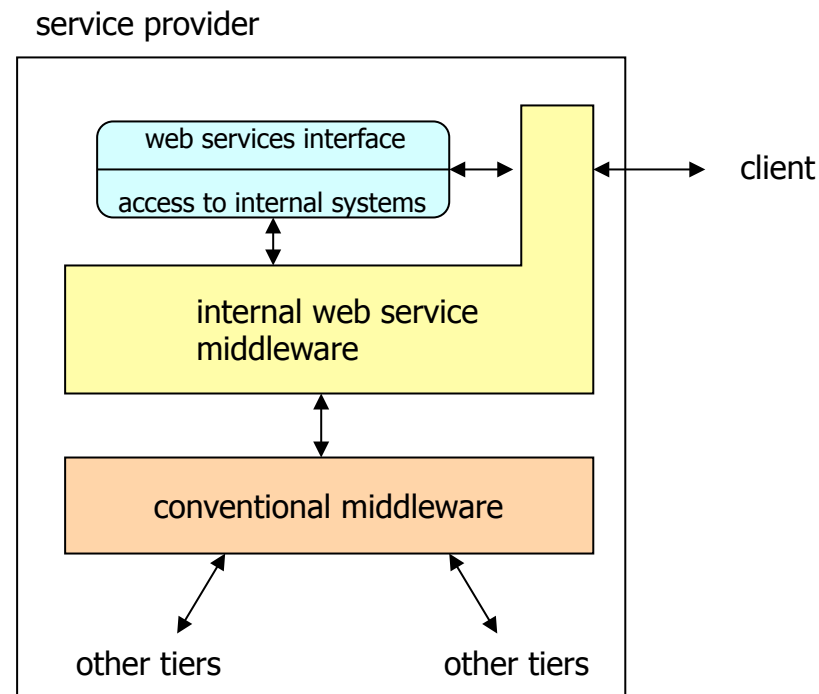
- W3C Web Services Architecture WG
 - produces WS Architecture Specification (working group note, 02/2004)
 - provide a common definition of a web service
 - define its place within a larger Web services framework to guide the community
- Definition
 - "A Web service is a **software system** designed to support **interoperable machine-to-machine interaction** over a network. It has an **interface** described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using **SOAP messages**, typically conveyed using **HTTP** with an **XML serialization** in conjunction with other **Web-related standards**."
 - Earlier, more general definition:
"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**."

(October 2002)

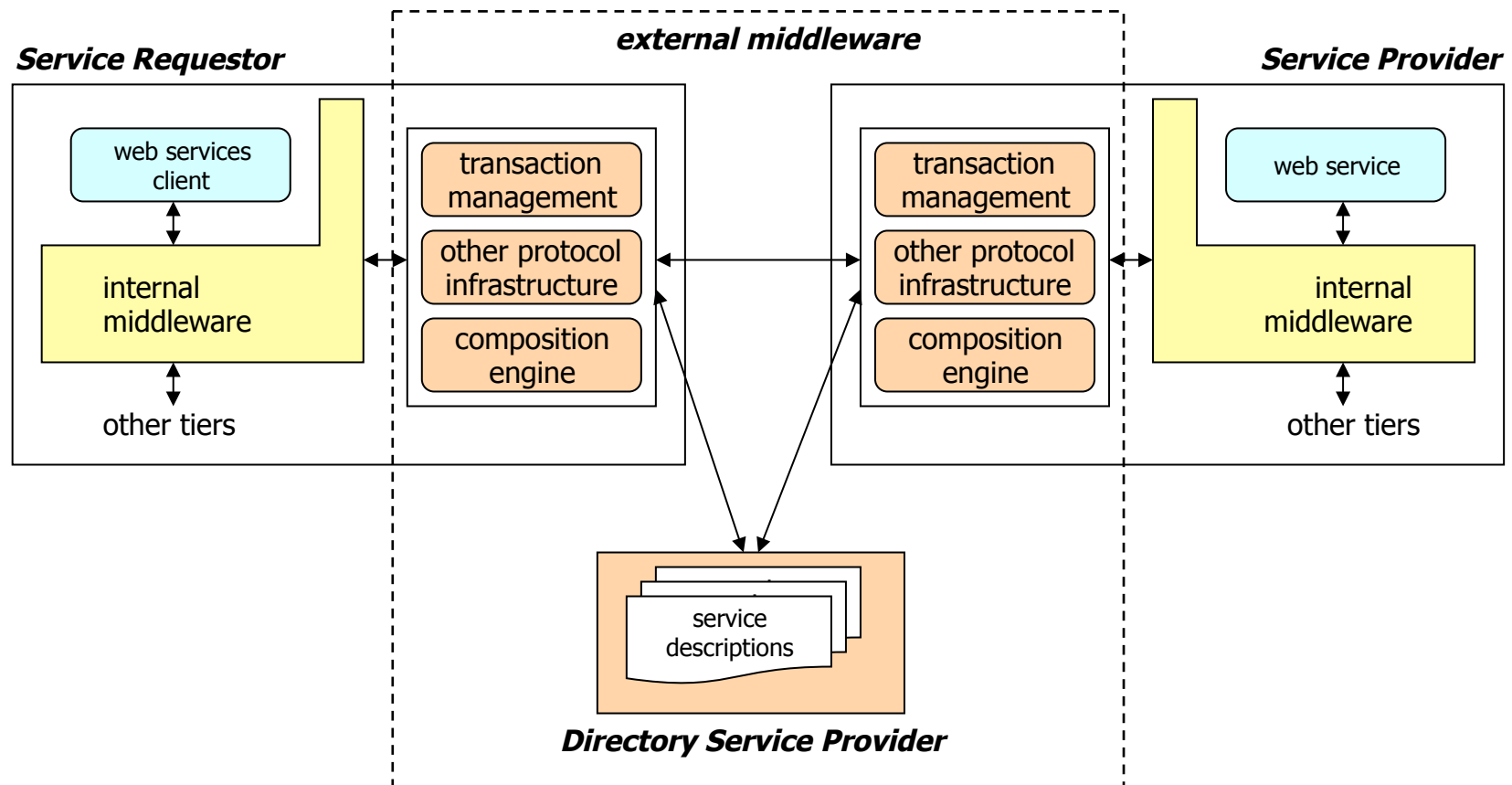


Web Service System Architecture

- Common internal architecture leveraging conventional middleware

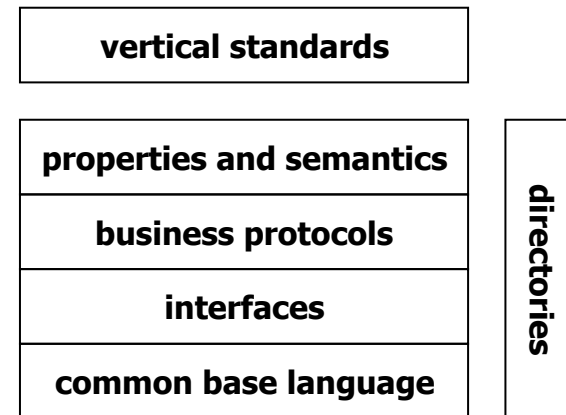


External Web Services Architecture



Technologies: Service Description & Discovery

- Service Description
 - Common Base Language (→XML)
 - Interfaces (→WSDL)
 - extend "traditional" IDLs
 - interaction mode
 - address/transport protocol info
 - Business Protocols (→WSCL, BPEL)
 - describe possible *conversations*
 - order of interactions
 - Properties and Semantics (→UDDI, WS-Policy)
 - descriptions to facilitate binding in a loosely-coupled, autonomous setting
 - e.g., non-functional properties (cost, transactional & security support)
 - textual descriptions
 - organize this information
 - Vertical Standards
 - interfaces, protocols, etc. specific to application domains



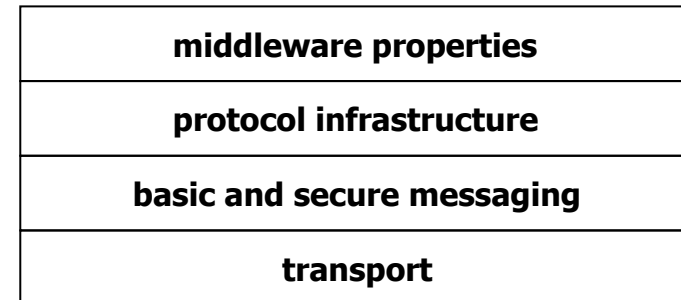
Service Description and Discovery Stack

- Service Discovery
 - Directory/Repository for WS descriptions
 - APIs and protocols for directory interaction
 - at design-time or run-time

Technologies: Service Interaction & Composition

■ Service Interaction

- Transport
 - lots of possibilities
 - HTTP most common
- Basic and Secure Messaging
 - standardize how to format/package information to be exchanged (→SOAP)
 - define how to extend basic mechanism to achieve additional capabilities (→WS-Security)
- Protocol Infrastructure (meta-protocols)
 - general infrastructure for business interactions
 - maintain state of conversation
 - meta-protocols
 - which protocols do we use?
 - who is coordinating?
- Middleware Properties (horizontal protocols)
 - properties similar to those of conventional middleware
 - reliability, transactions, ...



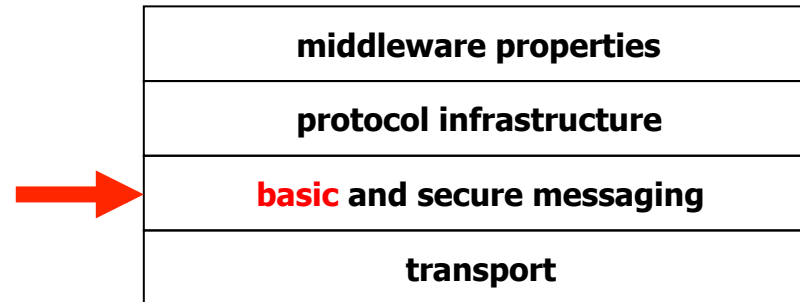
Service Interaction Stack

■ Service Composition

- Implement web service by invoking other web services
- Similar to workflow management, only for web services



SOAP – Simple Object Access Protocol



Service Interaction Stack

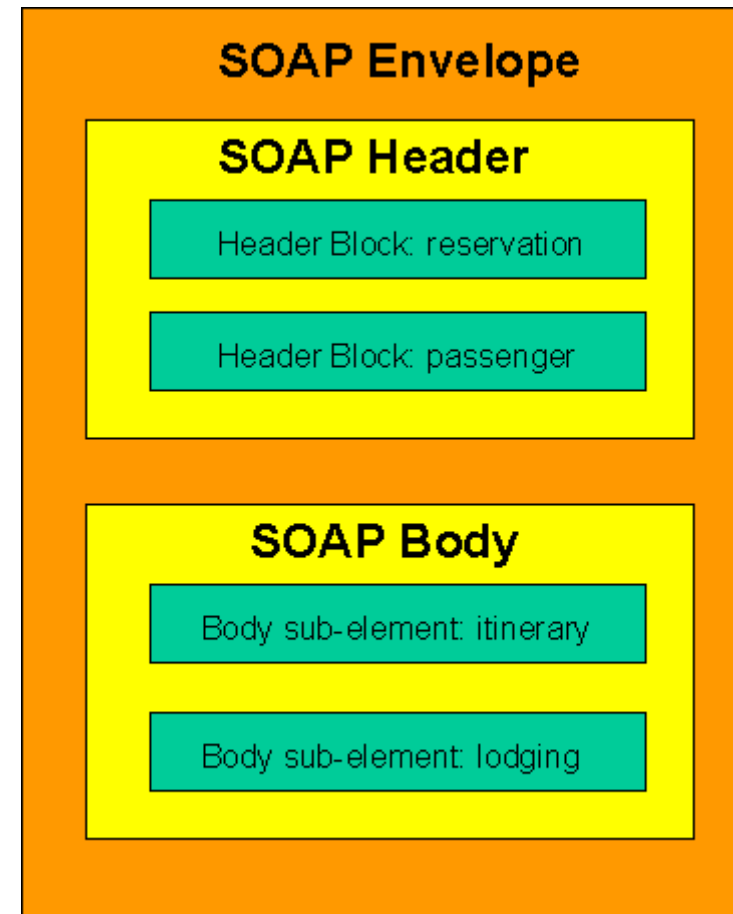
- Defines how to format information in XML so that it can be exchanged between peers
 - message format for **stateless, one-way communication**
 - support loosely-coupled applications
 - conventions for interaction patterns (RPC)
 - implement "on top of" one-way messaging
 - first message encodes the call, second (reply) message the result
 - processing rules for SOAP messages
 - how to transport SOAP messages on top of HTTP, SMTP

SOAP Envelope Framework

- Defines mechanism for identifying
 - What information is in the message
 - Who should deal with the information
 - Whether this is optional or mandatory
- **Envelope** element is the root element of the SOAP message, contains
 - Optional **header** element
 - Mandatory **body** element
- **Body** element
 - Contains arbitrary XML
 - application-specific
 - Child elements are called body entries (or bodies)
- Some consequences
 - Message body cannot contain general XML **document**, only elements
 - Validation of application data (i.e., the body) requires separation from the surrounding SOAP-specific XML
 - Many web service engines support that

Sample SOAP Message

```
<?xml version='1.0' ?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
      env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
      <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
    </m:reservation>
    <n:passenger xmlns:n="http://mycompany.example.com/employees"
      env:role="http://
      www.w3.org/2003/05/soap-envelope/role/next"
      env:mustUnderstand="true">
      <n:name>Áke Jógvan Øyvind</n:name>
    </n:passenger>
  </env:Header>
  <env:Body>
    <p:itinerary xmlns:p="http://travelcompany.example.org/reservation/travel">
      <p:departure>
        <p:departing>New York</p:departing>
        <p:arriving>Los Angeles</p:arriving>
        <p:departureDate>2001-12-14</p:departureDate>
      </p:departure>
      <p:return>
        <p:departing>Los Angeles</p:departing>
        <p:arriving>New York</p:arriving>
        <p:departureDate>2001-12-20</p:departureDate>
      </p:return>
    </p:itinerary>
    <q:lodging xmlns:q="http://travelcompany.example.org/reservation/hotels">
      <q:preference>none</q:preference>
    </q:lodging>
  </env:Body>
</env:Envelope>
```

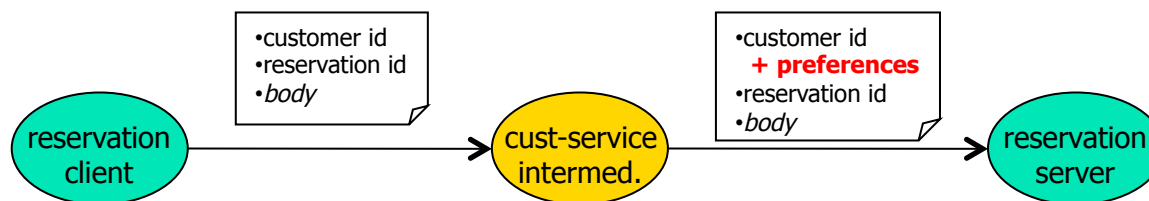


SOAP Headers

- Primary extensibility mechanism in SOAP
 - Additional facets can be added to SOAP-based protocols
 - Mechanism to
 - provide additional "control" information (e.g., directives, context information)
 - pass information that is orthogonal to the specific information to execute the request
 - Any number of headers can appear in a SOAP envelope
- Usage areas
 - Application-specific extensions (see previous example)
 - e.g., reservation identification, customer identification and information, ...
 - Generic service extensions
 - authentication, authorization, transaction management, payment processing, tracing, auditing
- Header content
 - Arbitrary XML
 - Determined by the schema of the header element

SOAP Intermediaries

- SOAP intermediaries provide "value-added services"
 - SOAP message can travel through multiple SOAP nodes
 - Sender [-> Intermediary ...] -> ultimate Receiver
 - Intermediaries process one or more SOAP headers
 - Header is removed from the message after processing (default behavior)
 - can be reinserted by the intermediary, possibly with modified values
 - Intermediary does not need to understand message body



- Relay attribute (optional)
 - relayable headers that were targeted at the intermediary but were not processed have to be forwarded
 - non-relayable headers that were targeted at the intermediary but were not processed have to be removed

SOAP Processing Model

- Describes logical actions taken by a node when receiving a SOAP message
- Every node has to
 - check message for syntactical correctness
 - analyze SOAP-specific parts
 - envelope, header, body elements
- Role attribute (optional)
 - governs further processing of header blocks
 - node assumes one or more roles, selects headers targeted at these roles
 - every node must assume the role "next"
 - predefined roles ("next", "ultimate_receiver", ...) vs. user-defined roles
- MustUnderstand attribute (optional)
 - if set to "true" for a selected header, a node assuming the target role **MUST** understand and be able to process it
 - generate fault if header cannot be processed, before any processing is started

SOAP-based RPCs

- SOAP is fundamentally a stateless, one-way message exchange paradigm
 - ...but applications can create more complex interaction patterns
 - Request/response, request/multiple responses
- SOAP-based RPC
 - Employs request/response message exchange pattern (MEP)
 - MEPs define "templates" for more complex message exchanges
 - Invocation is modeled as a struct of in/inout parameters
 - `<doCheck>`
 - `<product> ... </product>`
 - `<quantity> ... </quantity>`
 - `</doCheck>`
 - Response is modeled as a struct as well
 - `<doCheckResponse> ... </doCheckResponse>`
 - All data is passed by-value
 - Endpoint (address of target node) to be provided in a protocol binding-specific manner
- Protocol Bindings and RPC
 - RPC not predicated to any protocol binding
 - Binding to HTTP (synchronous protocol) makes RPC-style "natural"
 - One-way exchange will use simple acknowledgement as HTTP response

A Simple SOAP/HTTP RPC

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: application/soap+xml ;
charset="utf-8"
Content-Length: nnnn
```

Object Endpoint

```
<SOAP-ENV:Envelope
```

```
  xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
```

```
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
```

```
  <SOAP-ENV:Body>
```

```
    <m:GetLastTradePrice xmlns:m="Some-URI">
```

```
      <symbol>DIS</symbol>
```

```
    </m:GetLastTradePrice>
```

```
  </SOAP-ENV:Body>
```

```
</SOAP-ENV:Envelope>
```

Method Name


Input Parameter

A Simple SOAP Response

HTTP/1.1 200 OK
Content-Type: application/soap+xml;
charset="utf-8,,
Content-Length: nnnn

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
  <SOAP-ENV:Body>
    <m:GetLastTradePriceResponse xmlns:m="Some-URI">
      <Price>34.5</Price>
    </m:GetLastTradePriceResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Standard
Suffix



More SOAP

- SOAP protocol bindings
 - SOAP standard defines a binding to HTTP
 - SOAP is transport-independent, can be bound to any protocol type
 - E.g., SMTP, message queuing systems, ...
- SOAP with Attachments
 - XML isn't good at carrying non-XML things within it
 - Introduces an outer multipart MIME envelope
 - Root part is SOAP envelope
 - Other parts can be anything: XML, images, ...

Beyond SOAP – WS-Addressing

- Source and Destination information
 - SOAP does not define them as part of the message itself
 - relies on protocol-specific bindings
 - Example: SOAP/HTTP
 - endpoint reference is a URL encoded in the HTTP transport header
 - destination of the response is determined by the return transport address
 - Information might be lost
 - transport connection terminates (timeout)
 - message forwarded by an intermediary (e.g., a firewall)
 - Response always goes to sender
 - not possible to have response go somewhere else
- WS-Addressing
 - provides a mechanism to place the target, source and other important address information directly within the Web service message
 - decouples address information from any specific transport model
 - w3c recommendation

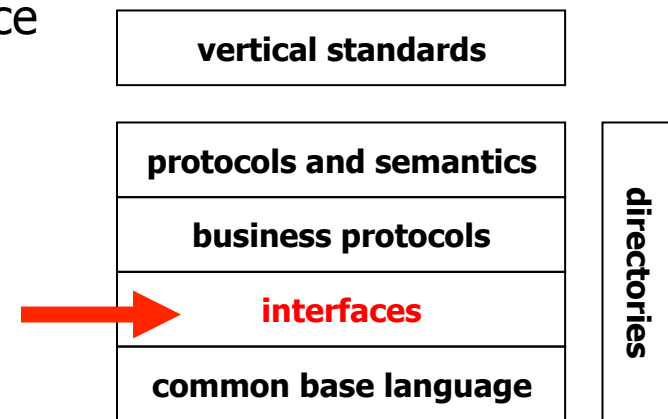
WS-Addressing Constructs

- Endpoint reference
 - uniquely identifies WS endpoint
- Message information headers
 - describe end-to-end message characteristics such as
 - source and destination endpoints
 - message identity
- Example

```
<S:Envelope xmlns:S="http://www.w3.org/2002/12/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <S:Header>
    <wsa:MessageID>
      http://example.com/6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://business456.com/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://fabrikam123.com/Purchasing</wsa:To>
    <wsa:Action>http://fabrikam123.com/SubmitPO</wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Web Services Description Language (WSDL)

- Provides all information necessary to programmatically access a service
 - documentation for distributed systems
 - recipe for automating the details involved in applications communication
- Description of the logical web service interface
 - operations, parameters, ...
 - similar to IDL in conventional middleware
- Describes mechanism to access the web service
 - which protocol is used
 - SOAP, ...
 - service location
- WSDL standardization pursued by w3c
 - V1.1 specification is a w3c note
 - not an official standard, but most widely used
 - WSDL 2.0 is a w3c recommendation

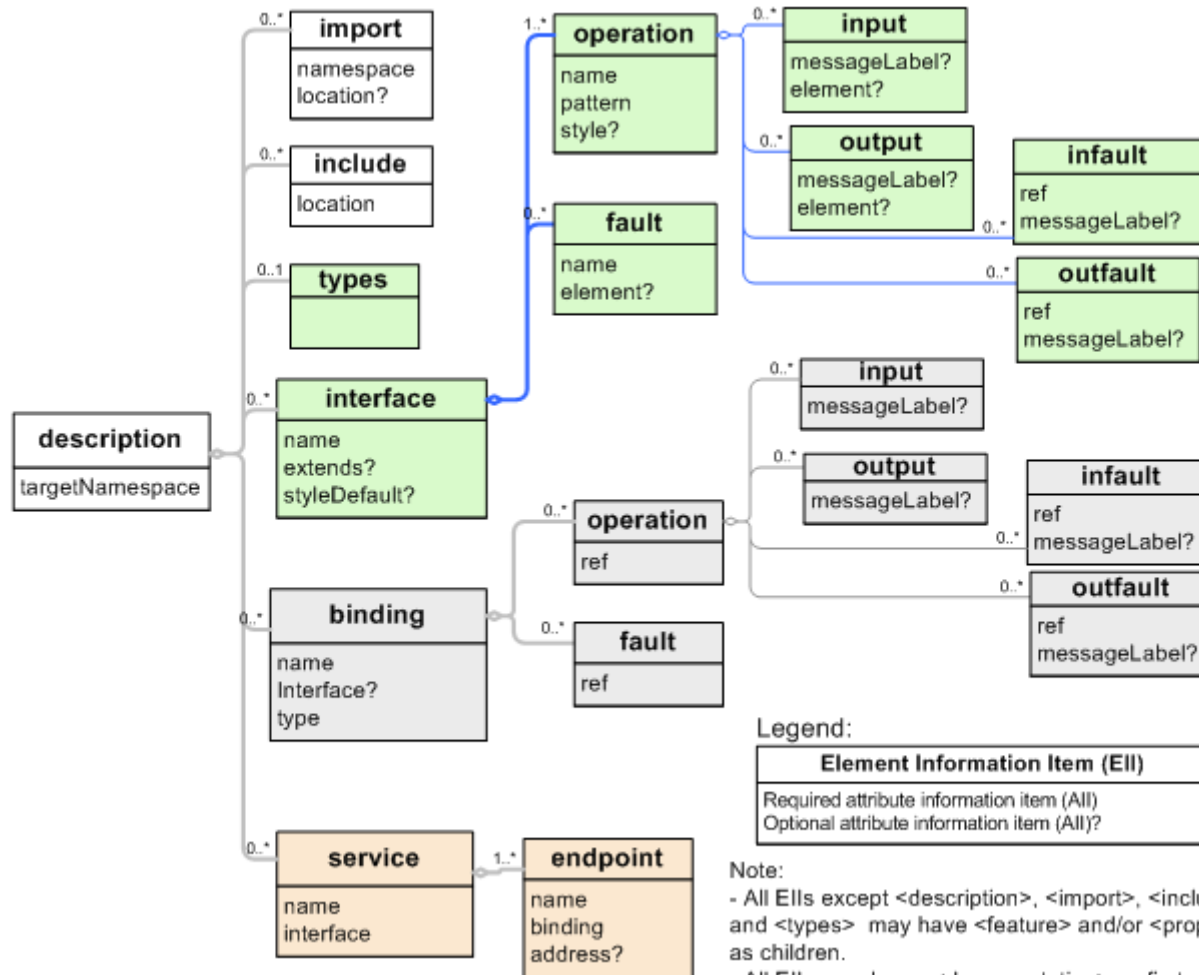


Service Description and Discovery Stack

Ingredients of WSDL

- Abstract part
 - Types: Definitions of data types needed
 - Message Exchange Pattern: Abstract definition of data exchanged
 - Operation: Abstract actions supported by the service
 - Interface: Interface defined as set of operations
- Concrete part
 - Binding: Concrete protocol and data format used to implement an interface
 - Endpoint: Single individual "end point" identified by a network address supporting a particular binding
 - Service: Collection of related "end points"

WSDL 2.0 Document Structure



Modularizing Service Definitions

- WSDL document defines a target namespace
 - similar to XML Schema target namespace
- Import/Include

```
<description>  
  [ <import namespace="uri" location="uri"/> | <include location="uri"/> ]*  
</description>
```
- Can be used to factor out any kind of definitions
 - Types, Interface, Bindings,... or any combination of these
 - Example:
 - Import Interface and specify Binding
 - Import Binding and specify Service
- Import, include differ regarding namespaces
 - include: referenced WSDL document needs to have same target namespace
 - import: referenced WSDL can have different target namespace
 - components are referenced in importing document using qualified names

Message Exchange Patterns

- Defines interaction paradigms
 - exchange of several asynchronous messages
 - sequence and cardinality of messages in an operation
 - abstract: no message types, no binding-specific information is specified
 - minimal contract
- Standard MEPs defined by WSDL specification
 - in-bound MEPs
 - In-Only, Robust In-Only, In-Out, In-Optional-Out
 - out-bound MEPs
 - Out-Only, Robust Out-Only, Out-In, Out-Optional-In
 - Where to send to? Outside scope of WSDL
 - Information could be provided through another (subscribe) operation or defined at deployment time
- Extensibility – possible to define new MEPs

Types

```
<description...>
```

```
  <types>
```

```
    <xsd:schema.../>*
```

```
  </types>
```

```
</description>
```

- Type clause used to define types used in message exchange
 - all message types (normal, fault) are single, top-level elements
- Default type system is XML Schema
 - Special extensibility element foreseen to refer to other type system
- Example

```
  <description targetNamespace= ...> ...
```

```
    <types>
```

```
      <xsd:schema ...>
```

```
        <xsd:complexType name="registration">
```

```
          ... </xsd:complexType>
```

```
        <xsd:element name="registrationRequest" type="registration"/>
```

```
      </xsd:schema>
```

```
    </types>
```

```
  ...
```



Interface

- Interface is a set of abstract operations
 - may extend other interfaces (i.e., multiple interface inheritance)
 - faults, operations, etc. are inherited
 - overloading of operations is not supported
 - inheritance conflicts must not occur
 - default style for operations can be specified
- Operation groups a set of abstract messages involved
 - references a MEP that defines sequence of messages
 - defines the structure of input, output, infault, outfault messages by referencing the appropriate (schema) types
 - optionally declares a style
 - rules used for generating messages, e.g., RPC style
 - may optionally be declared "safe"
 - no further obligations result from an invocation
- Interface Fault
 - definition of faults that can occur in the scope of this interface

Binding

- Interface, type elements define the abstract, reusable portion of the WSDL definition
- The binding element tells the service requestor **how to format the message in a protocol-specific manner**
 - interface can have one or more bindings
- Protocol-specific aspects are provided using binding extensions

```
<binding name="..." interface="..."?>  
  <!-- extensibility element (1) -->*  
  <operation ref="..."?>*  
    <!-- extensibility element (2) -->*  
    <input messageLabel="..."?>?  
      <!-- extensibility element (3) -->*  
    </input>  
    <output messageLabel="..."?>?  
      <!-- extensibility element (4) -->*  
    </output>  
    <infault ref="..." messageLabel="..."?>*  
      <!-- extensibility element (5) -->*  
    </infault>  
    <outfault ref="..." messageLabel="..."?>*  
      <!-- extensibility element (6) -->*  
    </outfault>  
  </operation>  
</binding>
```

- Standard binding extensions for SOAP/HTTP, HTTP GET/POST, SOAP w/MIME attachments

SOAP Binding - Details

- <soap:binding>
 - protocol: HTTP, SMTP, FTP, ...
 - mep: default SOAP message exchange pattern for operations
- <soap:operation>
 - action: value of SOAPAction HTTP header (SOAP over HTTP only!)
 - mep: actual mep for the operation
 - e.g., soap-response for implementing an in-out WSDL MEP

Endpoint and Service

- Endpoint
 - Specifies the network address of the endpoint hosting the web service
- Service
 - Contains a set of related endpoint elements
 - Group endpoints related to the same service interface but expressed by different protocols (bindings)
- Example

```
<service name="StockQuoteService"
  interface="StockQuoteInterface">
  <endpoint name="StockQuoteEndpoint"
    binding="tns:StockQuoteSoapBinding">
    <address="http://myservice.com/stockquote" />
  </port>
</service>
```

implemented binding

address of the endpoint

Web Service Policies

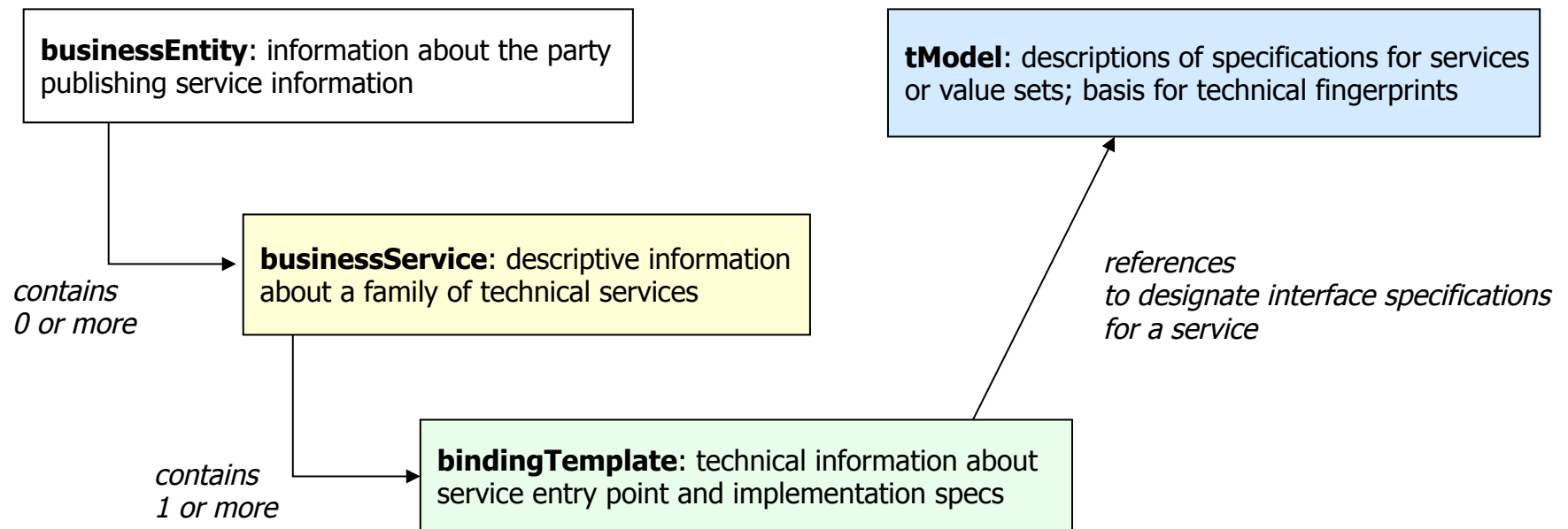
- Web service capabilities and requirements need to be described as (machine-readable) metadata
 - examples: addressing, security, transactions, reliability
 - allows tools to check for service compatibility, generate code
- WS-Policy
 - express capabilities, characteristics of entities in a WS-based system
 - policy **assertions**, **expressions**, statements
 - example:

```
<All>  
  <wsam:Addressing>...</wsam:Addressing>  
  <ExactlyOne>  
    <sp:TransportBinding>...</sp:TransportBinding>  
    <sp:AsymmetricBinding>...</sp:AsymmetricBinding>  
  </ExactlyOne>  
</All>
```
 - allows senders, receivers to specify their security requirements and capabilities
- WS-PolicyAttachment
 - associate policy expressions with subjects
 - reference policies from WSDL definitions or inline them in bindings
 - associate policies with UDDI entities

Universal Description Discovery and Integration (UDDI)

- Goal: enable service discovery
 - catalogue services based on published information of service providers
 - maintain taxonomy(ies) to support searching for appropriate services in business terms
 - specify technical binding information to actually communicate with the selected service
- UDDI registry serves as a directory of web services
 - Allows searching “by what” and “by how” instead of just “by name”
- UDDI defines
 - Set of schemas for describing businesses and their services
 - UDDI data model
 - SOAP API for accessing a UDDI registry
- UDDI initiative
 - Involves more than 300 companies
 - <http://www.uddi.org>

UDDI Core Data Structures



- UDDI key
 - uniquely identifies each instance of core data structures within a registry
 - basis for realizing the containment/referencing relationships (using foreign keys)
- XML Schema definition for UDDI Data Model

Important Registry APIs

- Inquiry API
 - Find things
 - find_business
 - find_service
 - find_binding
 - find_tModel
 - Get Details about things
 - get_businessDetail
 - get_serviceDetail
 - get_bindingDetail
 - get_tModelDetail
- Publishers API
 - Save things
 - save_business
 - save_service
 - save_binding
 - save_tModel
 - Delete things
 - delete_business
 - delete_service
 - delete_binding
 - delete_tModel
 - security...
 - get_authToken
 - discard_authToken

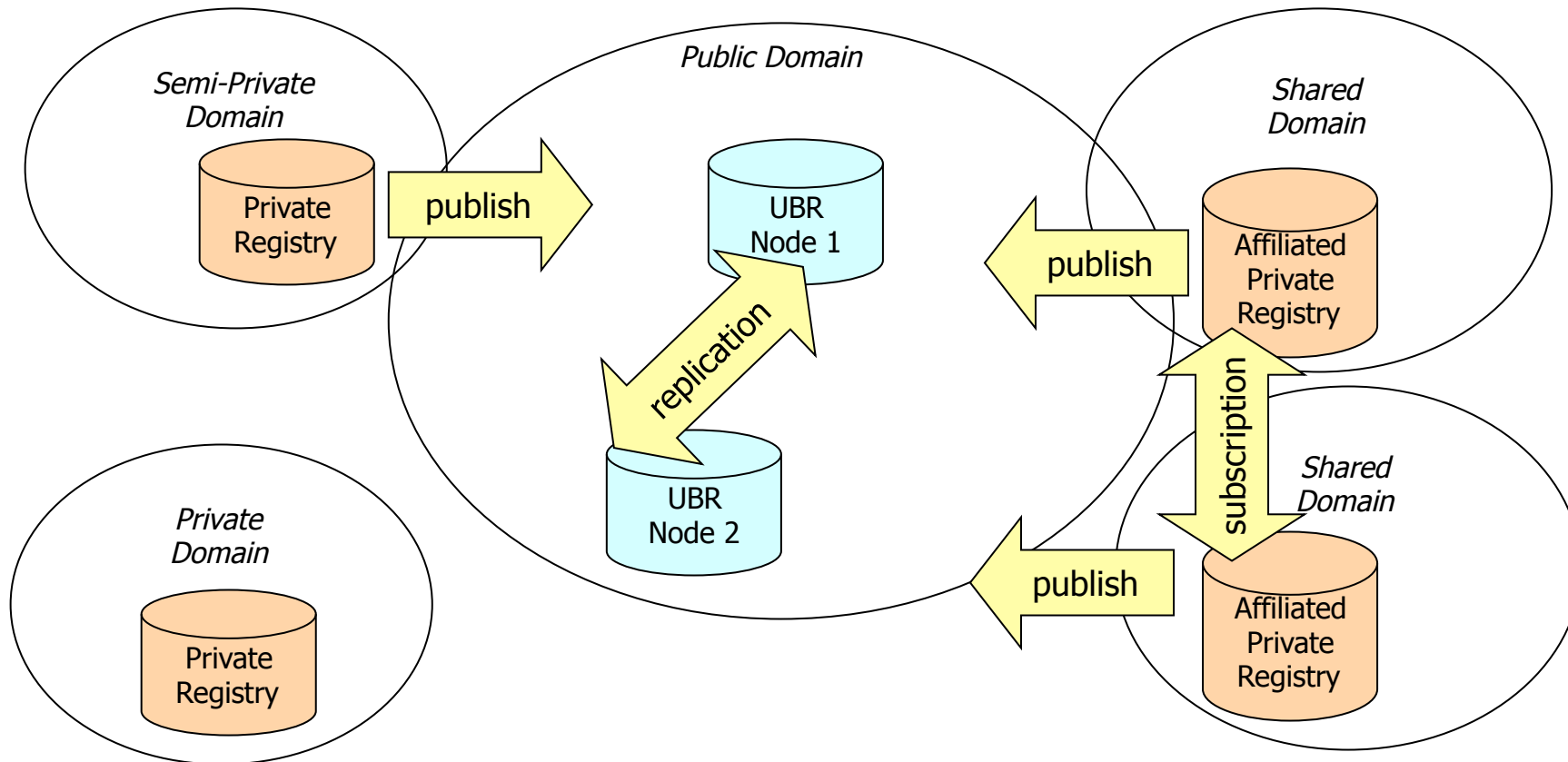
Provided as SOAP-based web services



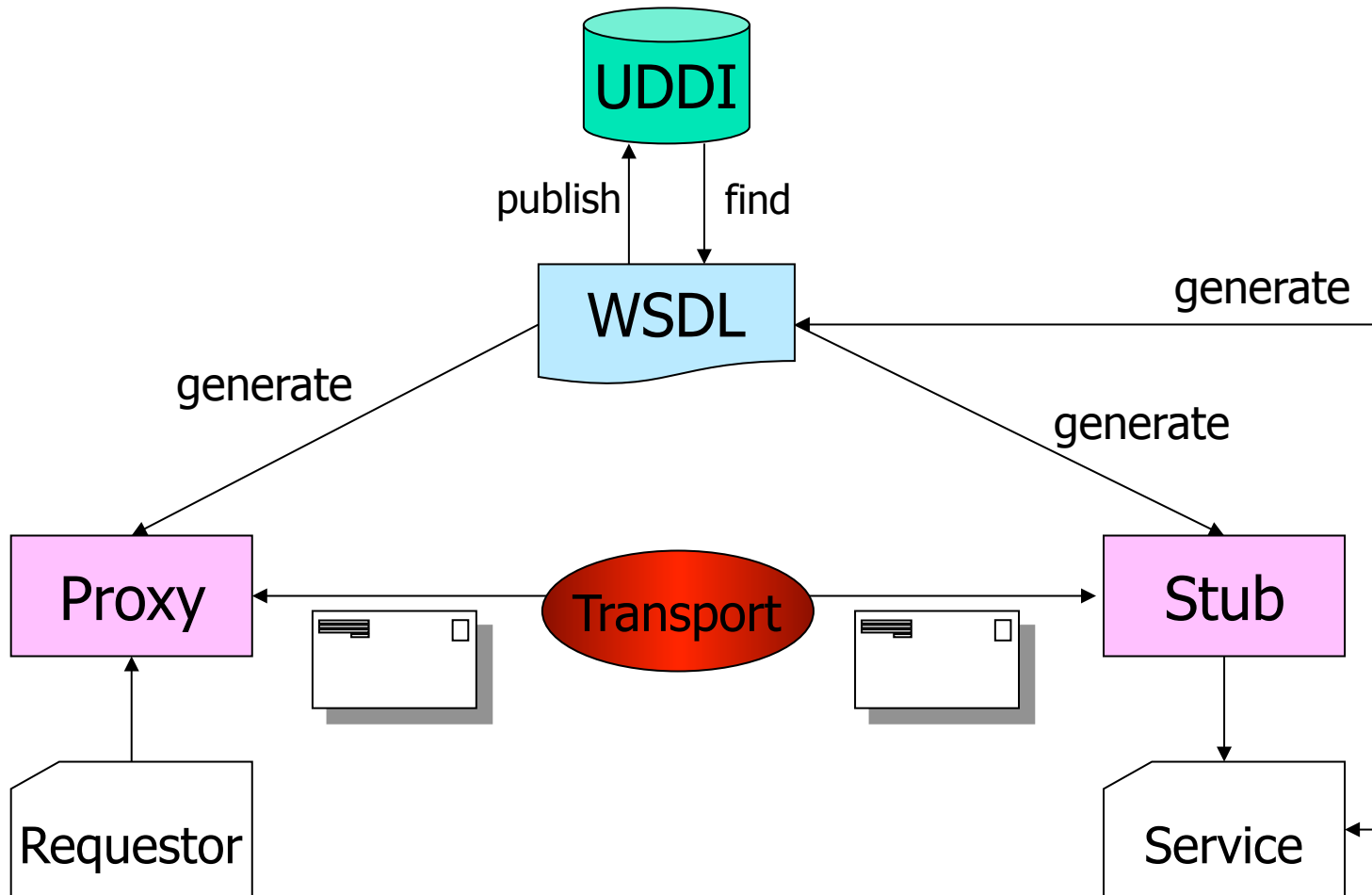
Registry Types

- Different types of registries
 - corporate/private (e.g., enterprise web service registry)
 - operates within the boundaries of a single company (or for a restricted number of partners)
 - data is not shared with other registries
 - affiliated (e.g., trading partner network)
 - registry is deployed in a controlled environment
 - limited access by authorized clients
 - data may be shared with other registries in a controlled manner
 - public (e.g., UDDI Business Registry)
 - open, public access to registry data
 - secured administrative access, content may be moderated
 - data may shared, transferred among registries
- UDDI Business Registry
 - public, global registry of businesses and their services
 - master directory of publicly available e-commerce services
 - was initial focus of UDDI effort

Registry Affiliation – Example



Tooling Principles



Java API for XML Web Services (JAX-WS)

- API for building web services and clients based on remote procedure calls and XML
 - Goal: hide all the complexities of SOAP message processing
 - APIs for supporting XML based RPC for the Java platform
 - Define web service
 - Use web service
 - Defines
 - WSDL/XML to Java mapping
 - Java to XML/WSDL mapping
 - Core APIs
 - SOAP support (including attachments)
 - Client and Server Programming models involving generated stub classes
- Client side invocation (standard programming model)
 - Application invokes web service through generated stub class
 - JAX-WS runtime maps the invocation to SOAP, builds the SOAP message, processes the HTTP request
- Server side processing
 - JAX-WS runtime processes HTTP, SOAP message, maps to RPC and dispatches to target (class implementing the web service)

Mapping WSDL <-> Java – Example

WSDL 1.1 interface definition:

```
<!-- WSDL Extract -->
<message name="getLastTradePrice">
  <part name="tickerSymbol"
    type="xsd:string"/>
</message>
<message
  name="getLastTradePriceResponse">
  <part name="result"
    type="xsd:float"/>
</message>
<portType
  name="StockQuoteProvider">
  <operation name="getLastTradePrice"
    parameterOrder="tickerSymbol">
    <input message=
      "tns:getLastTradePrice"/>
    <output message=
      "tns:getLastTradePriceResponse"/>
  </operation>
</portType>
```

Java service endpoint interface:

```
//Java
public interface StockQuoteProvider
  extends java.rmi.Remote {
  float getLastTradePrice(
    String tickerSymbol)
    throws java.rmi.RemoteException;
}
```



REST

- REST (Representational State Transfer)
 - architectural style for building large-scale distributed hypermedia systems
 - see dissertation by Roy Thomas Fielding (2000)
- Four main principles
 - resource identification through URI
 - uniform interface
 - HTTP GET, PUT, POST, DELETE
 - self-descriptive messages
 - resources are decoupled from their representation
 - content can be accessed in a variety of formats (content negotiation)
 - stateful interactions through hyperlinks
 - no session state maintained on the server
 - explicit state transfer using self-contained messages
- Basis for scalability of the web
 - caching, clustering, load balancing

RESTful Web Services

- Web services provided purely based on the above principles
 - alternative to WSDL/SOAP-based „big“ web services
- Rationale
 - perceived to be simple
 - leverages existing, well-known standards (HTTP, XML, URI, MIME)
 - light-weight infrastructure that requires only minimal tooling, is inexpensive
 - similar to building dynamic web site
 - REST and Ajax (see later chapter) complement each other nicely
- Drawbacks and limitations
 - (still) needs careful design and enumeration of resources to be exposed, mapped to generic interface, and of data representations used
 - more flexible, but more format variations to account for
 - only RPC-style interactions, HTTP-only
 - requires manual implementation of reliability, transactions

Summary

- Service-oriented architectures
 - definition, access, discovery of (web) services
- SOAP
 - defines SOAP message structure and messaging framework
 - stateless, one-way
 - more complex patterns "on top" (e.g., request/response)
 - provides convention for doing RPCs using SOAP
 - support for extensibility, error-handling, flexible data representation
 - independent of transport protocols
 - binding framework for defining protocol-specific bindings
 - SOAP/HTTP
 - extensions beyond SOAP for addressing, reliable messaging (see next chapter)

Summary (cont.)

- WSDL
 - supports description of all information needed to access a web service
 - interface, operation, message types
 - binding to specific protocol (e.g., SOAP)
 - protocol extensions
 - endpoint, service
- UDDI
 - registry
 - publish information about business, services provided, and the way to use them
 - white, yellow, green pages
 - tModels provide infrastructure for business and service "name space"
 - identification, classification of business, services, protocols, ...
 - can "point to" detailed service descriptions such as WSDL files
 - APIs for manipulating and inquiring about registry content
 - provided as web services

Summary (cont.)

- Application development
 - Integration with programming languages, existing middleware
 - Tooling support
- Programming language binding
 - WSDL as the "IDL for web services"
 - Mapping WSDL to PL (e.g., Java)
 - enables generation of client proxies, server stubs for web services invocation
 - Mapping PL to WSDL
 - "publish" existing functionality as a web service
 - Example: JAX-RPC
- Web services support based on conventional middleware
 - define standards for reusing/extending existing programming models and middleware infrastructure to support web service
 - Java EE: use/publish servlets, stateless session beans to implement web services
- REST (Representational State Transfer)
 - idea: model a set of services as resources with generic interface (get, put, post, delete)
 - "lightweight", stateless web services