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

Technische Universität Kaiserslautern

Chapter 14 – Foundations of Web Services

Motivation & Introduction
Basic Web Services Technology
Web Services Support in Middleware Platforms

What's a Web Service?

- "A Web Service is programmable application logic accessible using standard Internet protocols..."
  Microsoft
- "A Web Service is an interface that describes a collection of operations that are network accessible through standardized XML messaging ..."
  IBM
- "Web services are software components that can be spontaneously discovered, combined, and recombined to provide a solution to the user's problem/request. The Java language and XML are the prominent technologies for Web services"
  Sun
- "A Web Service is a 'virtual component' that hides 'middleware ideosynchracies' like the underlying component model, invocation protocol, etc. as far as possible"
  Frank Leymann (IBM)

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)*

Types of E-Business

Business To Consumer (B2C)	**Business To Business (B2B)**	**Intra Business**
• Relation between enterprise and customers • Sales-related aspects are predominant, like product presentation, advertising, service advisory, shopping	• Relation between processes of different enterprises • Predominant are relation to suppliers, and customer relations to other enterprises like industrial consumers, retailers, banks	• Electronic organization of internal business processes, like realization within workflow systems

B2B Integration – Conventional Middleware

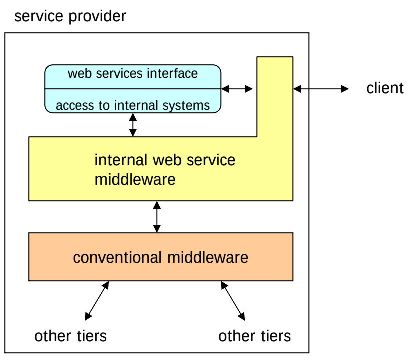
- Middleware itself is (logically) centralized
  - usually controlled by a single company
  - now requires agreement on using, managing specific middleware platform across companies ("third party")
  - need to implement a "global workflow"
  - problems
    - lack of trust
    - autonomy needs to be preserved
    - 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

Web Service System Architecture

- Common internal architecture leveraging conventional middleware

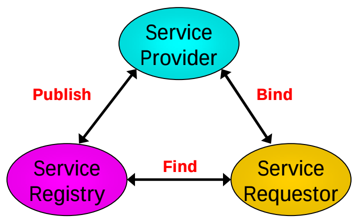


Service-Oriented Architecture (SOA)

- 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"
- Principal concepts
  - service – mechanism to enable access to one or more capabilities
    - provider and consumer roles
    - service opaqueness
      - invocation interface, separate from implementation
  - service-based interactions involve
    - visibility (awareness, willingness, reachability)
      - availability of service descriptions and policies
    - interaction
      - interaction modes
      - information model – characterizes information exchange (syntax, semantics)
      - behavior model – action model, process model
    - real world effect
      - return information and/or change some shared state
  - service description
  - policies and contracts – contrain the service use, reach service use agreement
  - service execution context

Service-Oriented Architecture (SOA)

- Service Requestor
  - Finds required services via Service Broker
  - Binds to services via Service Provider
- Service Provider
  - Provides e-business services
  - Publishes availability of these services through a registry
- Service Registry
  - Provides support for publishing and locating services
  - Like telephone yellow pages

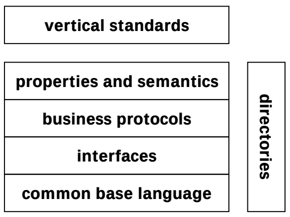


Granularity of Services

- Services can be simple and composite
  - check credit card number
  - raise a mortgage
- Simple services are…
  - …provided as servlets, EJBs, Assembler programs,…
- Composite services are…
  - …provided via "choreography"
    - Referring to other fine grained services
    - Scripting fine grained services into business processes
  - via workflow technology

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

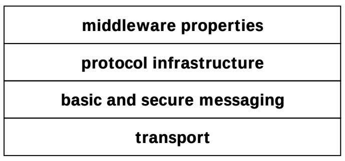


Service Description and Discovery Stack

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

Technologies: Service Interaction & Composition

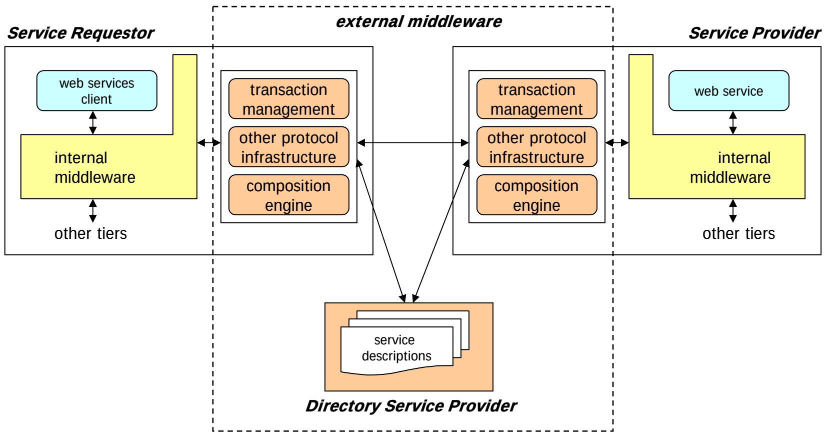
- Service Interaction
  - Transport
    - lots of possibilities
    - HTTP most common
  - Basic and Secure Messaging
    - standardize how format/package information to be exchanged (→SOAP)
    - define how to extend basic mechanism to achieve additional capabilities (→WS-Security)
  - Protocol Infrastructure (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

External Web Services Architecture



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

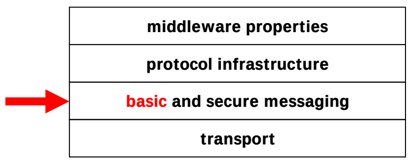
Technische Universität Kaiserslautern

Basic Web Services Technology

Standards

- XML (eXtensible Markup Language)
  - Underlying basic representation approach (common syntax)
- SOAP (Simple Object Access Protocol)
  - Standardized interaction
    - common data format
    - conventions for different forms of interaction (messaging, RPC)
    - bindings to lower-level transport protocols (HTTP, SMTP)
  - Messages (not RPCs) as the basic communication unit
    - loose coupling, broad range of supported protocols
- WSDL (Web Services Description Language)
  - Description of a service's programming interface
  - XML-based interface definition language
- UDDI (Universal Description, Discovery and Integration)
  - Registry of and search for web services information
    - equivalent of a naming and directory service in conventional middleware

SOAP – Simple Object Access Protocol

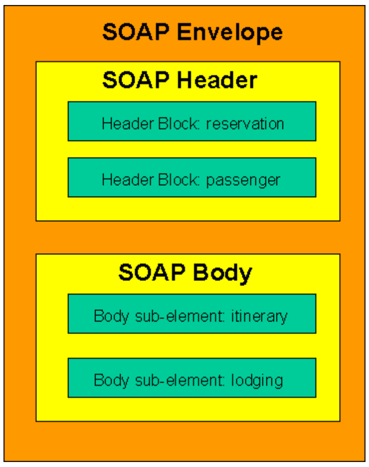


Service Interaction Stack

- 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

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>Åke 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 Envelope
SOAP Header
Header Block: reservation
Header Block: passenger
SOAP Body
Body sub-element: itinerary
Body sub-element: lodging

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 elements
  - 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

SOAP Headers

- Primary extensibility mechanism in SOAP
  - Additional facets can be added to SOAP-based protocols
  - Mechanism to 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
  - Authentication, authorization, transaction management, payment processing, tracing, auditing
- Header content
  - Arbitrary XML
  - Determined by the schema of the header element
- Processing of a header by recipient may be
  - Mandatory (attribute mustUnderstand=“true”)
  - Optional

SOAP Header - Example

```
<SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
  <SOAP-ENV:Header>
   <t:Transaction xmlns:t="some-URI, SOAP-ENV:mustUnderstand="true">
        5
   </t:Transaction>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
   <m:GetLastTradePrice xmlns:m="Some-URI">
        <symbol>DEF</symbol>
   </m:GetLastTradePrice>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Protocol Extension

SOAP Processing Model Terminology

- sender
  - Node that transmits a SOAP message.
- receiver
  - Node that accepts a SOAP message.
- message path
  - Set of SOAP nodes through which a single SOAP message passes. This includes the initial SOAP sender, zero or more SOAP intermediaries, and an ultimate SOAP receiver.
- initial sender
  - Sender that originates a SOAP message at the starting point of a SOAP message path.
- intermediary
  - Both a receiver and a sender. Targetable from within a SOAP message. Processes the SOAP header blocks targeted at it and acts to forward a SOAP message towards an ultimate receiver.
- ultimate receiver
  - Final destination of a SOAP message. Responsible for processing the contents of the SOAP body and any SOAP header blocks targeted at it. Cannot also be an intermediary for the same SOAP message

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
  - predefined roles ("next", "ultimate_receiver", ...) vs. user-defined roles
- MustUnderstand attribute (optional)
  - if set to "true" for a selected header, node MUST understand and be able to process it
    - generate fault if header cannot be processed, before any processing is started

SOAP Intermediaries

- SOAP intermediaries
  - 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
    - Example: separate authentication/authorization from service implementation
    - 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

Error Handling in SOAP

- SOAP Fault element
  - Returned as the single element inside the body of the response
- Fault element indicates which error occurred and provides diagnostic information through child elements
  - *Code* element (required)
    - Hierarchical namespace of faultcode values
      - E.g., Client.AuthenticationFailure
    - Top level codes:
      - VersionMismatch
      - MustUnderstand – a required header was not understood
      - Client – likely cause is content or formatting of the SOAP message
      - Server
  - *Reason* element contains human-readable message
- Ability to signal a fault depends on the underlying messag transfer mechanism
  - protocol binding has to specify the details

SOAP Data Encoding

- Encoding simple data types (e.g., strings, integers, booleans, …) is easy
  - Use the corresponding XML Schema representation
  - The xsi:type can be used to further describe the data type passed in the message
    - Example:

```
<SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
                    <symbol xsi:type="xsd:string">DEF</symbol>
        </m:GetLastTradePrice>
 </SOAP-ENV:Body>
```

- For more complex types (e.g., arrays, arbitrary objects), one may want to use a specific encoding
  - Attribute **encodingStyle** can appear in any element in a SOAP message
- SOAP defines set of encoding rules, based on XML Schema
  - SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/
    - SOAP arrays, structures, …
  - Usage is not mandatory
    - E.g., a vendor may support an optimized encoding format

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

<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>
```

Object Endpoint

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 candidate 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>
```

Beyond SOAP – WS-ReliableMessaging

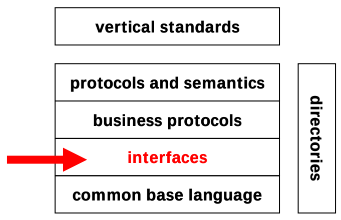
- Goal: enable Web services to ensure delivery of messages over unreliable communication networks
  - simplifies application development
    - support by runtime/tooling vendors
  - allows existing message-oriented middleware to interoperate through a common protocol
- WS-ReliableMessaging specification
  - creates a modular mechanism for reliable message delivery
  - defines a messaging protocol to identify, track, and manage the reliable delivery of messages between a source and a destination
  - defines a SOAP binding (required for interoperability)
  - can be combined with other WS "components"
    - WS-Security, WS-Policy, ...
  - industry specification published by BEA, IBM, Microsoft, and Tibco

Reliable Messaging Delivery Assurances

- Application-level delivery guarantees for a (sequence of) message(s)
  - *AtMostOnce:* Message duplication is avoided. It is possible that some messages in a sequence may not be delivered.
  - *AtLeastOnce:* Guaranteed delivery. Some messages may be delivered more than once.
  - *ExactlyOnce:* Every message sent will be delivered without duplication.
    - logical "and" of the two prior delivery assurances
  - *InOrder:* Messages will be delivered in the order that they were sent. This delivery assurance may be combined with any of the above delivery assurances.
- Assurances are guaranteed by the endpoints implementing the WS-ReliableMessaging protocol
  - responsibility lies with initial sender/ultimate receiver
  - supported by the protocol
- Each message sequence can have its own delivery assurance
  - uses "policy attachments" for web services

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
- WSDL specification
  - standardization pursued by w3c
    - http://www.w3.org/TR/wsdl
  - V1.1 specification is a w3c note
    - not an official standard, but most widely used
  - WSDL 2.0 is a w3c candidate recommendation



Service Description and Discovery Stack

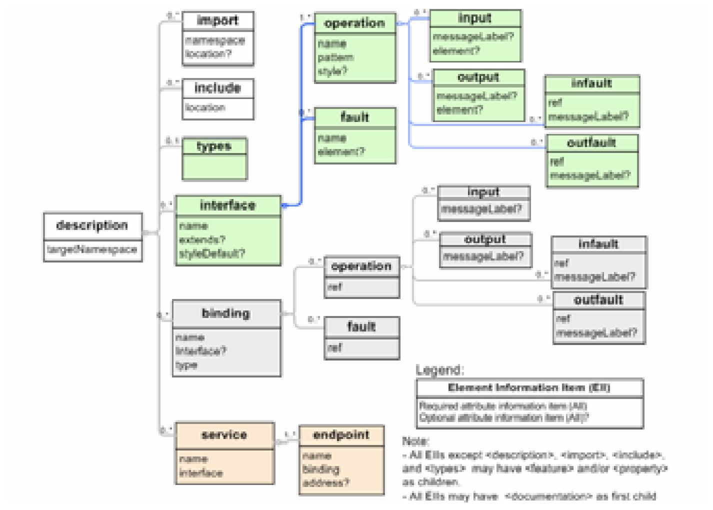
WSDL Goals

- Provides a description of the logical interface of a web service
  - operations, parameters, ...
  - similar to IDL in conventional middleware
- Also describes mechanism to access the web service
  - which protocol is used
    - SOAP, ...
  - service location
- Support modular specifications
  - same service interface can be provided through different protocols and data formats, at different locations
- Defines interaction paradigms (message exchange patterns)
  - exchange of several asynchronous messages

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

- Define sequence and cardinality of messages in an operation
  - abstract: not 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
  - fault model
    - *robust*, *-optional-**: fault message may be sent as a reply
    - *In-Out, Out-In*: fault message may replace a reply
    - **-Only*: do not generate fault messages
- 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

Interface Syntax (Simplified)

```
<description targetNamespace="xs:anyURI" >
. . .
    <interface name="xs:NCName" extends="list of xs:QName"?
                styleDefault="list of xs:anyURI"? >
        <fault name="xs:NCName" element="xs:QName"? > </fault>*
        <operation name="xs:NCName" pattern="xs:anyURI" style="list of xs:anyURI"?
                wsdlx:safe="xs:boolean"? >
            <input messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? > </input>*
            <output messageLabel="xs:NCName"? element="union of xs:QName, xs:Token"? > </output>*
            <infault ref="xs:QName" messageLabel="xs:NCName"? > </infault>*
            <outfault ref="xs:QName" messageLabel="xs:NCName"? > </outfault>*
        </operation>*
    </interface>*
    . . .
</description>
```

RPC Style

- Designed to facilitate programming language bindings to WSDL
  - ensure that the messages can be mapped to function/method signatures
- Can be used in combination with MEPs in-only, in-out
- Message schemas have to follow the following rules
  - structure of input/output messages is defined as complex type with sequence
  - no complex content models (e.g., choice, group, ...) allowed with sequence
  - only local elements allowed as sequence items (but may be nillable, have multiple occurrence)
  - local name of input message element corresponds to the operation name
  - local name of output message element is a concatenation of operation name | "Response"
  - no attributes allowed for content model of input/output messages
  - ...

Example

```
. . .
<types>
 <xs:element name="checkAvailability">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="checkInDate"
                 type="xs:date"/>
         <xs:element name="checkOutDate"
                 type="xs:date"/>
         <xs:element name="roomType"
                 type="xs:string"/>
       </xs:sequence>
     </xs:complexType>
 </xs:element>
 <xs:element name="checkAvailabilityResponse">
     <xs:complexType>
       <xs:sequence>
         <xs:element name="roomType"
                 type="xs:string"/>
         <xs:element name="rateType"
                 type="xs:string"/>
         <xs:element name="rate"
                 type="xs:double"/>
       </xs:sequence>
     </xs:complexType>
 </xs:element> ...
</types>
```

```
<interface name = "reservationInterface" >
  <operation name="checkAvailability"
       pattern="http://www.w3.org/2006/01/wsdl/in-out"
       style="http://www.w3.org/2006/01/wsdl/rpc"
       wrpc:signature= "checkInDate #in checkOutDate
       #in roomType #inout rateType #out rate #return">
    <input messageLabel="In"
       element="tns:checkAvailability" />
    <output messageLabel="Out"
       element="tns:checkAvailabilityResponse" />
  </operation>
   . . .
</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)
    - Group related but different interfaces together
- Example

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

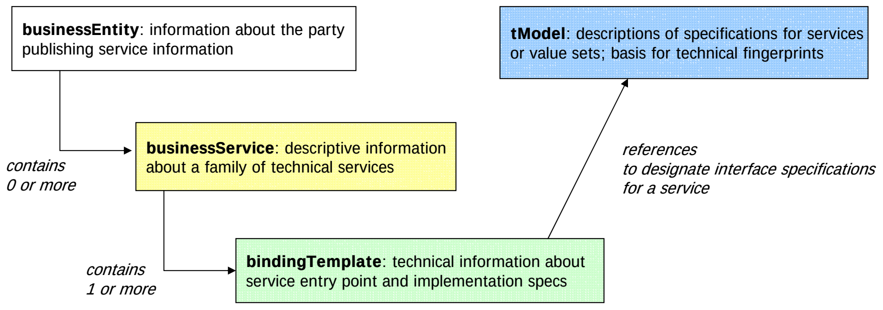
implemented binding

address of the endpoint

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


businessEntity: information about the party publishing service information
tModel: descriptions of specifications for services or value sets; basis for technical fingerprints
businessService: descriptive information about a family of technical services
contains
0 or more
references
to designate interface specifications for a service
bindingTemplate: technical information about service entry point and implementation specs
contains
1 or more

- 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

BusinessEntity

- *Business* key: UDDI key
- Descriptive information about the business entity offering services
  - (multiple) name(s) and textual description(s), possibly in multiple languages
  - contact info
    - names, phone numbers, e-mail addresses, postal addresses, descriptions
  - known identifiers
    - list of identifiers that a business may be known by, in different identifier systems
      - tax number, D-U-N-S, ...
  - business categories describing specific business aspects
    - categorization by industry, product, geographic region, ...
  - discovery URLs referring to other documents or resources describing the business entity
- Business services, describing families of web services offered

BusinessService

- *Services* key: UDDI key
- *Business* key: identifies the provider of the service
- Information describing a logical service in business (not technical) terms
  - (multiple) name(s) and textual description(s), possibly in multiple languages
  - business categories describing the provided service (see businessEntity categories)
    - categorization by industry, product, geographic region, ...
- Binding templates providing technical descriptions of the web services constituting the business service
  - e.g., the set of web services implementing a logical financial service

BindingTemplate

- *Binding* Key: UDDI key
- *Service* Key: identifies the logical service implemented by the web service
- Information businesses an instance of a web service offered at a particular network address
  - (multiple) textual description(s), possibly in multiple languages
  - access point representing the network address (e.g., URL) for invoking the service
  - categories describing specific aspects of the service
- tModelInstanceDetails
  - points to one or more tModel information elements
  - goal: provide a technical "fingerprint" for identifying compatible services

What Are tModels?

- A tModel (technology model) represents a concept, an idea, a well accepted technical specification (taxonomy, interface...)...
  - Its semantics should be clearly described
  - UDDI comes with a set of predefined tModels
- Examples
  - Taxonomies
    - NAICS (industry codes), UNSPC (product & service codes), ISO3166 (geographic locations) ...
  - Technical specifications
    - RosettaNet, ebXML, EDI, standard ERP system interface,...
  - Identifiers
    - D&B numbers, US tax codes,...
- When registering a tModel it gets a globally unique identifier: tModelKey
- tModel data structure
  - tModelKey, name, overviewDoc, descriptions, categories, identifiers, ...
    - overviewDoc may contain a URL child element that points to a WSDL file describing the interface ...

Using tModelKeys

- tModelKey is used to give references a semantics

```
<element name = "keyedReference">
  <type content = "empty">
   <attribute name = "tModelKey" type = "string"/>
   <attribute name = "keyName" minOccurs = "1" type = "string"/>
   <attribute name = "keyValue" minOccurs = "1" type = "string"/>
  </type>
</element>
```

- This allows to specify the semantics of a name-value pair, e.g.: Is the identifier a US Tax Number, is it D&B number, is the name of an interface of the system of a particular ERP vendor,...?
  - Example: identify SAP AG by its Dun & Bradstreet D-U-N-S® Number, using the corresponding tModelKey within the UDDI Business Registry

```
<keyedReference
   tModelKey=”uddi:ubr.uddi.org:identifier:dnb.com:D-U-N-S”
   keyName=”SAP AG”
   keyValue=”31-626-8655” />
```

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

Inquiry API

- FIND APIs
  - Basic browsing/searching
    - Can return a set of results
  - Limited search capabilities
    - Query is specified in an XML element with subelements for
      - Values of properties to match (e.g., business name starts with 'S')
      - Qualifiers that modify the search behavior (e.g., exactNameMatch, sortByNameDesc, ...)
    - Example: Find the latest two businesses that registered, and whose name starts with an 'S'

      -
        ```
        <find_business generic="1.0" maxRows="2" xmlns="urn:uddi-org:api">
            <findQualifiers>
                <findQualifier>sortByDateDesc</findQualifier>
            </findQualifiers>
            <name>S</name>
        </find_business>
        ```
  - Return unique reference keys identifying the result "elements"
- GET APIs
  - Based on unique reference keys, retrieve detailed information

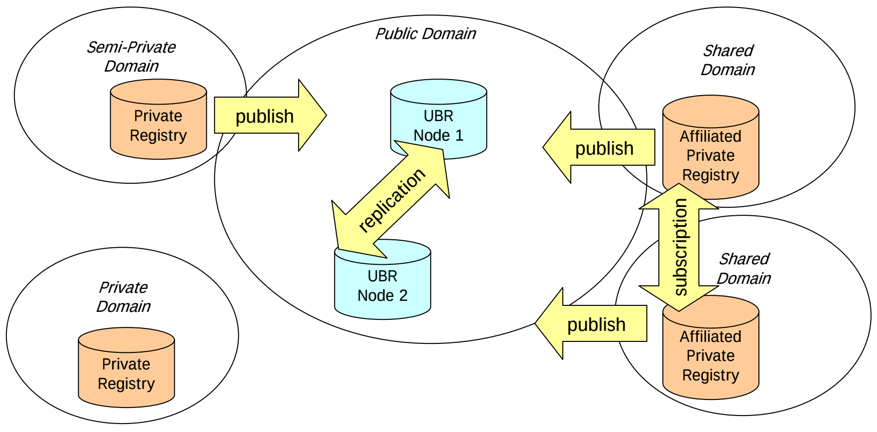
Registry Types

- 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 Architecture

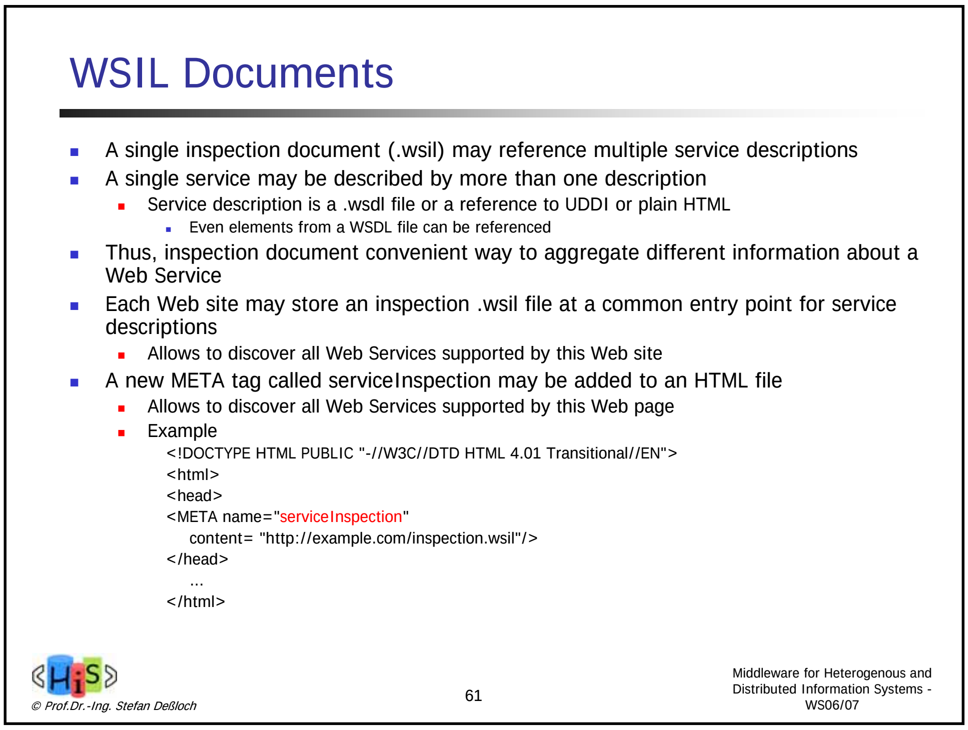
- UDDI registry may consist of multiple UDDI nodes
- UDDI node
  - supports interaction with UDDI data through (subset of) UDDI APIs
  - belongs to exactly one UDDI registry
  - interacts with other nodes in the same registry (through replication) to maintain a single, complete logical copy of the registry data
- Affiliation of registries
  - consists of multiple registries
  - registries define policies for controlled copying of subsets of registry data among each other
  - registries share a common namespace for UDDI keys, have compatible policies for assigning key values
- Enhanced set of APIs to support registry architecture, types of registries
  - security, custody transfer, subscription, replication

Registry Affiliation – Example



Discovering Web Services – Without UDDI

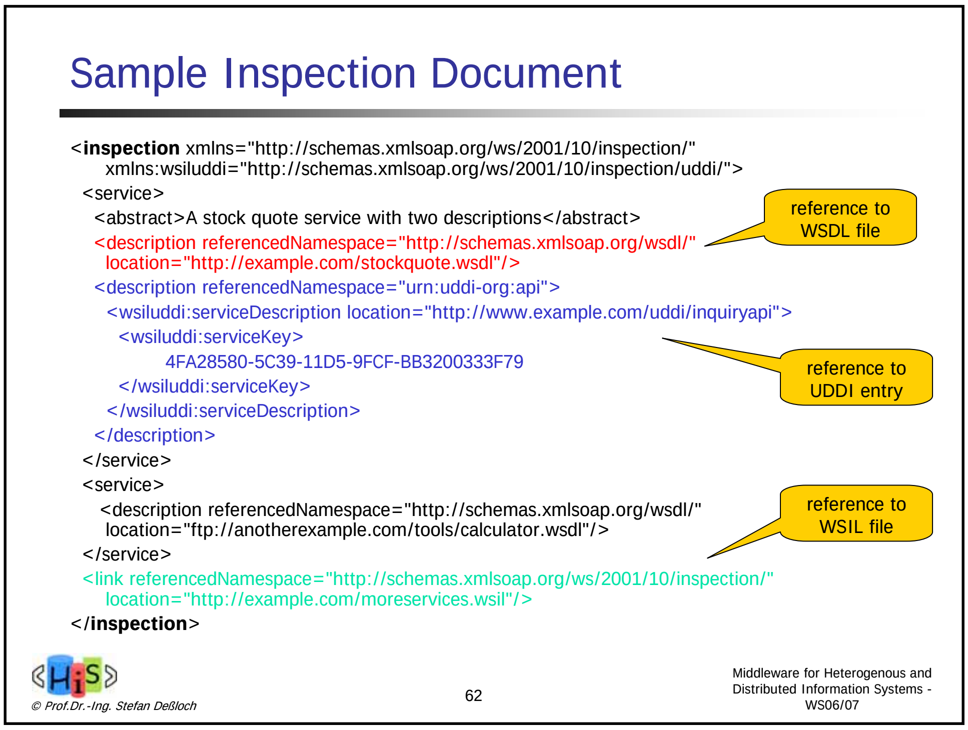
- Sometimes you don't want to register a Web Service in UDDI (yet)
  - It may not be of public interest
  - It may not be ready for production
  - …
- Web Services Inspection Language (WSIL)
  - Language to discover Web Services at Web sites
    - document-based, decentralized approach for web services discovery
  - Proposed by IBM and Microsoft (11/2001)
  - Supported by toolkits
    - Apache's Axis project
    - ...



WSIL Documents

- A single inspection document (.wsil) may reference multiple service descriptions
- A single service may be described by more than one description
  - Service description is a .wsdl file or a reference to UDDI or plain HTML
    - Even elements from a WSDL file can be referenced
- Thus, inspection document convenient way to aggregate different information about a Web Service
- Each Web site may store an inspection .wsil file at a common entry point for service descriptions
  - Allows to discover all Web Services supported by this Web site
- A new META tag called serviceInspection may be added to an HTML file
  - Allows to discover all Web Services supported by this Web page
  - Example

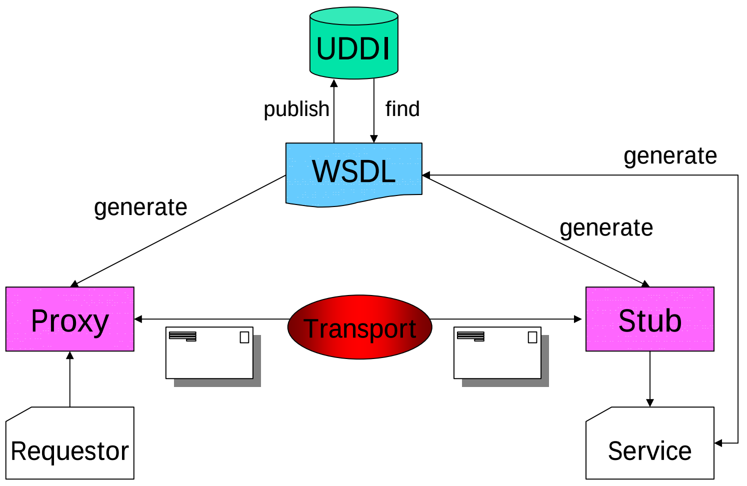
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<META name="serviceInspection"
    content= "http://example.com/inspection.wsil"/>
</head>
    ...
</html>
```



Sample Inspection Document

```
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
   xmlns:wsiluddi="http://schemas.xmlsoap.org/ws/2001/10/inspection/uddi/">
 <service>
  <abstract>A stock quote service with two descriptions</abstract>
  <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
   location="http://example.com/stockquote.wsdl"/>
  <description referencedNamespace="urn:uddi-org:api">
   <wsiluddi:serviceDescription location="http://www.example.com/uddi/inquiryapi">
    <wsiluddi:serviceKey>
        4FA28580-5C39-11D5-9FCF-BB3200333F79
    </wsiluddi:serviceKey>
   </wsiluddi:serviceDescription>
  </description>
 </service>
 <service>
   <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
   location="ftp://anotherexample.com/tools/calculator.wsdl"/>
 </service>
 <link referencedNamespace="http://schemas.xmlsoap.org/ws/2001/10/inspection/"
   location="http://example.com/moreservices.wsil"/>
</inspection>
```

Tooling Principles



Java API for XML-based RPCs (JAX-RPC)

- 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-RPC runtime maps the invocation to SOAP, builds the SOAP message, processes the HTTP request
- Server side processing
  - JAX-RPC 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>
```

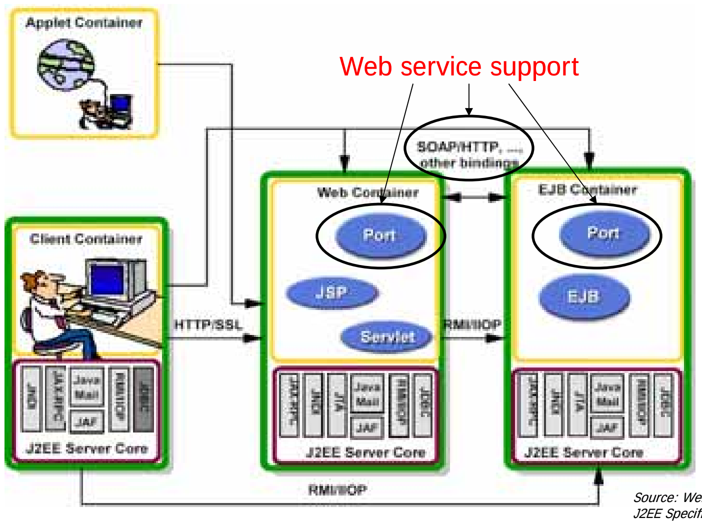
- Java service endpoint interface

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

Web Services for J2EE Specification (WS4J2EE)

- Sun specification (JSR109), included in J2EE 1.4
- Objectives (among others)
  - Simple model for defining and implementing a new Web service and deploying this into a J2EE application server
  - Build on evolving industry standards (WSDL, SOAP, ...)
  - Leverage existing J2EE technology
  - Inter-operability of vendor implementations
  - Minimize new concepts, interfaces, file formats, etc.
- WS4J2EE requires JAX-RPC support

J2EE Architecture



Source: Web services for J2EE Specification 1.0

Creating a Web Service

- Steps
  - Define service endpoint
    - Option 1: Start with WSDL, generate Java endpoint interface
    - Option 2: Start with Java endpoint interface, generate WSDL
  - Implement the service endpoint interface
    - J2EE Component Model
      - stateless session bean
      - servlet
    - allows for component reuse
  - Deploy the service on a server-side container-based runtime
    - specific to the runtime, deployment tool
    - deployment tool
      - configures one or more protocol bindings for the (abstract) service endpoint
        - e.g., SOAP/HTTP
      - creates one or more (concrete) endpoints with endpoint address
      - export the WSDL describing the service, so that clients can use it

Client Programming Model

- Client can be
  - J2EE application client
  - Web component
  - EJB component
  - Another web service
- Client view of web service
  - Set of methods that perform business logic
    - Service endpoint interface
  - Stateless, i.e., there is not state information that persists across method invocations
- Uses the WS4J2EE runtime to access and invoke the methods of a web service
  - JNDI lookup to access a Service object
    - Factory to obtain a stub/proxy that implements the service endpoint interface
  - Invoke web service method on the stub object implementing the service endpoint interface
- Client developer does NOT generate stub/proxy class during development
  - Will be generated during deployment of the client component
  - Can be specific to the vendor runtime used on the client

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

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
  - alternative for finding web services: WSIL

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
  - J2EE: use/publish servlets, stateless session beans to implement web services
    - JAX-RPC and SAAJ APIs
      - basic web services interoperability support
    - Web Services for J2EE specification
      - describes the packaging and deployment requirements for J2EE applications that provide and use web services
    - EJB specification
      - extended to support implementing web services using stateless session beans.
    - JAXR API
      - access to registries and repositories.
    - JAXP API
      - processing XML documents
        - Java interfaces to XSLT, SAX, DOM-parsers