Web Services Foundations

Worflows und Web Services
Kapitel 3

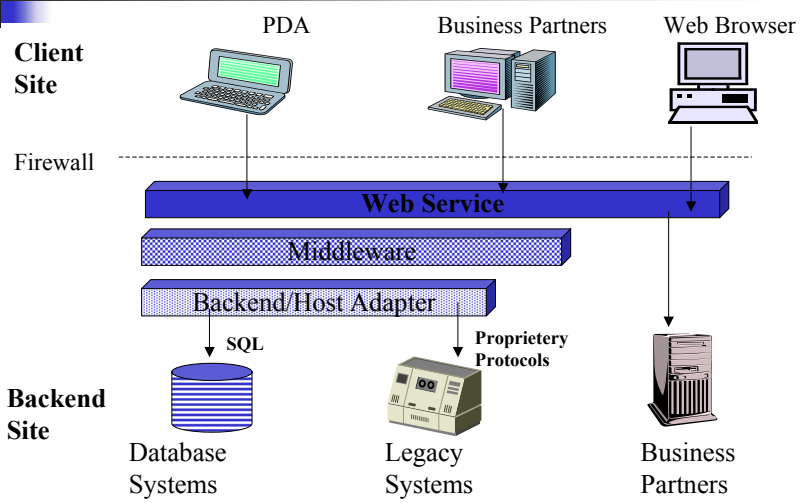
Web Services Evolution – Organizing Software Granules

- Subroutines, Functions
  - Centered around functional decomposition
  - Allows a system to be modularized, i.e. subdivided
  - Results into concept of APIs
- Objects
  - Centered around combining functions and data into encapsulated units
  - Enables concepts of classes, inheritance, polymorphism
  - Results into practice of building class lattices
- Services
  - Centered around making functions available on the Web
  - Enables dynamic e-business
  - Results into organizing services into taxonomies

What's a Web Service?

- "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."
  W3C Web Services Architecture Requirements, Oct. 2002
- "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

Web Service System Architecture

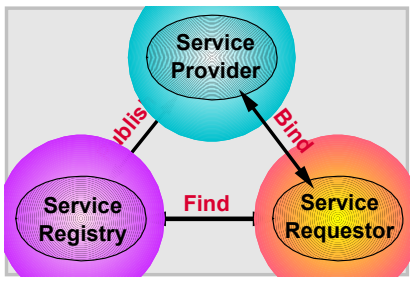


Software Aspect: Web Services

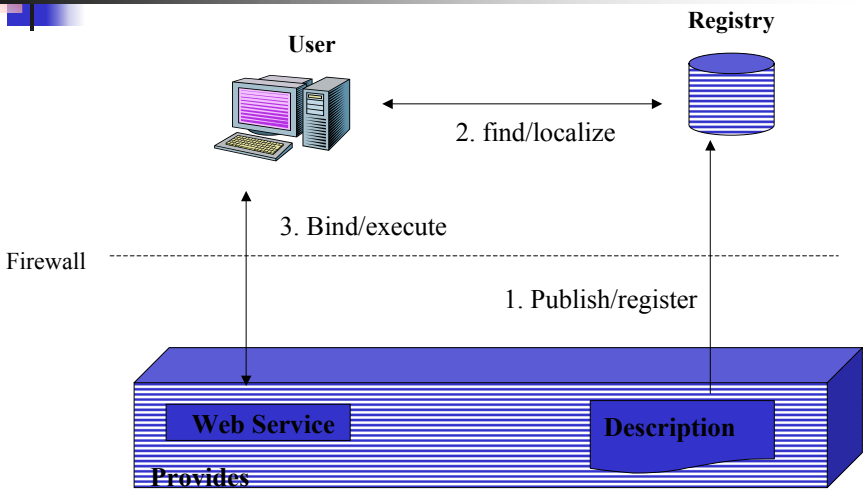
- A web service is a piece of software made available on the Web
- An architecture is service based iff it focuses on
  - formats and protocols for communication between services
    - E.g. RosettaNet, OBI,...
- An architecture is service oriented iff it focuses on
  - How to support the dynamic discovery of appropriate services at runtime: Find!
  - How services are organized: Understand!
  - How services are described: Invoke!

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



Web Service Model



Granularity of Services

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

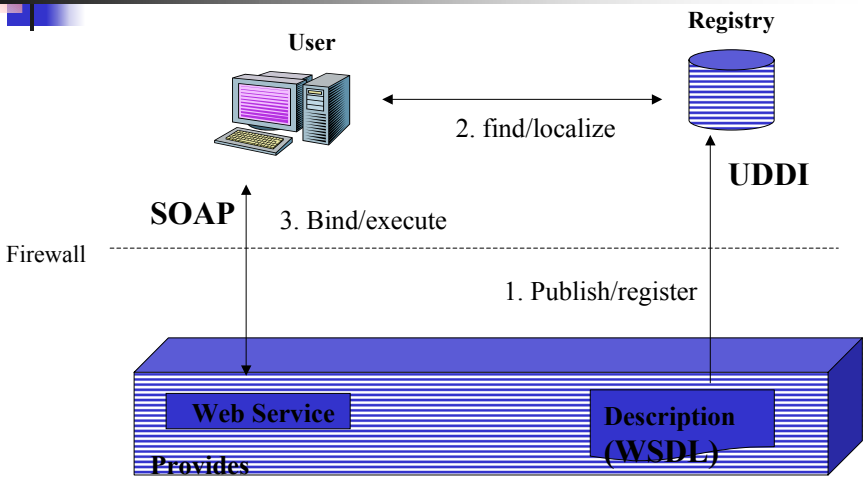
Class Hierarchies vs. Taxonomies

- Class lattice/library stimulates reuse
  - Eases communication between designers within a given community (development team,...)
- Class lattice/library captures syntax (signature) mostly
  - semantics „known" within community
- Services must be organized to capture semantics because no predefined community exists that reasons about services (community = „world"!)
  - Just signatures do not allow to capture semantics
  - Human beings have to capture semantics too
- Service taxonomies used to capture syntax and semantics of services

Standards

- UDDI
  - Universal Description, Discovery and Integration
  - Registry of and search for web services
  - Predefined schemas
- SOAP
  - Simple Object Access Protocol
  - Communication protocol
- WSDL
  - Web Services Description Language
  - Description of a service's functionality
- XML
  - eXtensible Markup Language
  - Underlying basic representation approach

Web Service Model (cont.)



Web Services Foundations - SOAP

(Object) RPCs Today

- Existing technologies like CORBA/IIOP, DCOM, or EJB are strong for server-to-server communication
  - ...but rely on closely administered environment
    - E.g. very cumbersome to allow two random computers to call each other out-of-the-box
- Weaknesses in client-to-server communication
  - Clients scattered across the Internet
    - Scalability, manageability, firewall,...
  - Very cumbersome to be used in Internet scenarios
  - Firewalls or proxy servers typically separate Internet clients from servers within the firewall

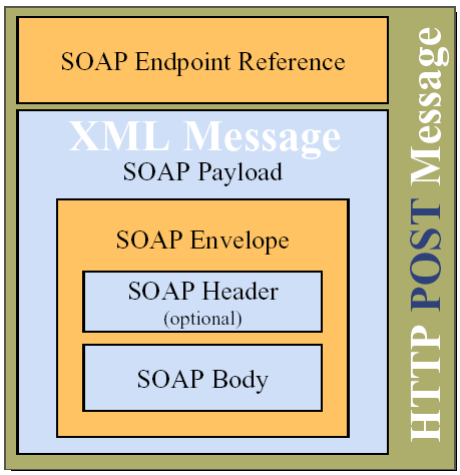
How to Access Web Services

- Firewalls are obstructions to ubiquitous access to services
- Wide range of different programming languages, programming paradigms and hosting environments,... makes orchestration a nightmare
- ... We need a globally available invocation mechanism!
- SOAP: Simple Object Access Protocol
  - Use of existing technology
    - HTTP & XML ("XML as HTTP payload")
      - HTTP as RPC transport
      - XML as RPC encoding scheme
  - No special SOAP API, no special SOAP ORB

Detour: What is HTTP?

- HTTP is a simple request/response protocol
- Client: Program that submits a request
  - GET /MyPath/MyHomePage.html HTTP/1.1
    Host: www.myserver.com
    Accept: text/html
    User-Agent: IE 5.5
  - POST /myFunctions/reverse HTTP/1.1
    Host: www.myserver.com
    Content-Type: text/plain
    Content-Length: 12
    Hello, World
- Server: Program that processes a request and returns a response (if applicable)
  - HTTP/1.1 200 OK
    Content-Type: text/html
    <HEAD>
    <TITLE>My Homepage</TITLE>
    </HEAD>
- No state is maintained between request/response pairs
- Being based on TCP, HTTP is reliable and connection oriented
- Can easily reach across firewalls

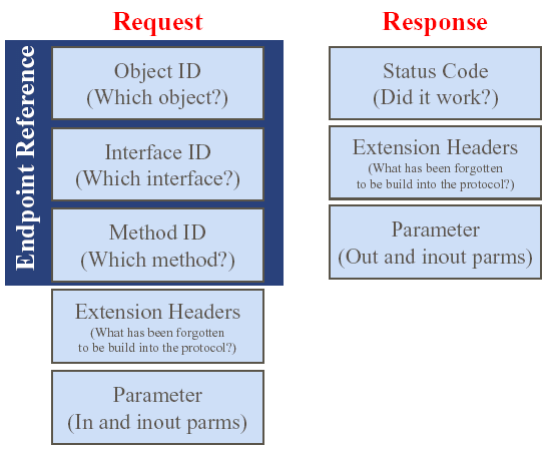
SOAP over HTTP



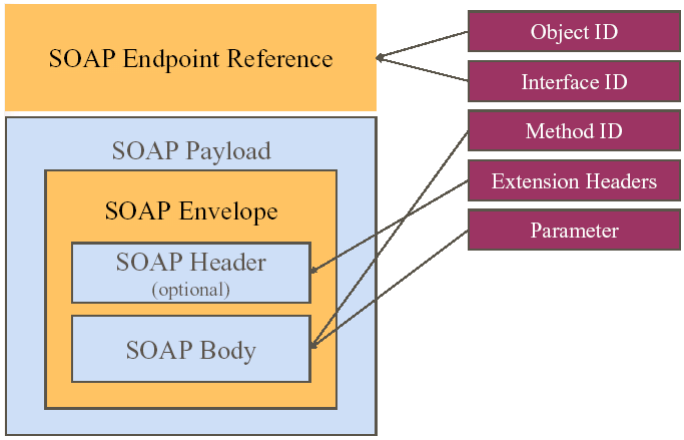
What Does SOAP Provide?

- SOAP Message as unit of communication
  - Envelope, body, headers
- SOAP Fault mechanism for error handling
- Extensibility mechanisms
  - XML, schemas, namespaces
  - SOAP headers for protocol extensions
- Flexible mechanism for data representation
- RPC mechanism
  - Calls and responses represented as SOAP messages
- Document-centric approach
  - Alternative to fine-grained RPC interfaces
- Binding to HTTP

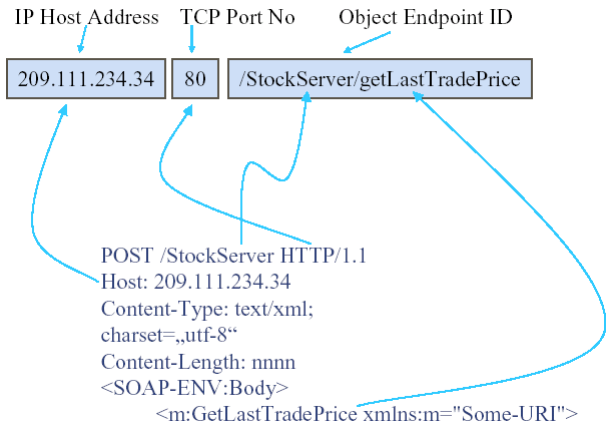
ORPC Request/Response



RPC via SOAP



SOAP Endpoint Reference



A Simple SOAP RPC

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/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: text/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

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 XML **document**, only elements
  - Validation of application data 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="1")
  - Optional

SOAP Header - Example

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml;
charset="utf-8„
Content-Length: nnnn
SOAPAction: „Some-URI"

<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="1">
          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

Error Handling in SOAP

- Communicated at two levels
  - HTTP response code
    - 500 Internal Server Error
  - 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
  - Faultcode 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
  - Faultstring element contains human-readable message
  - Faultactor element: where in the message path did the fault occur?

Failing to Honor Mandatory Header

HTTP/1.1 500 Internal Server
ErrorContent-Type: text/xml;
charset="utf-8„
Content-Length: nnnn

```
<SOAP-ENV:Envelope
xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
          <faultcode>SOAP-ENV:MustUnderstand</faultcode>
          <faultstring>SOAP Must Understand Error</faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


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>
```

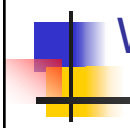
- For more complex types (e.g., arrays, arbitrary objects), one may want to use a specific encoding
  - Attribute **encodingStyle** can appear in any element in a SOAP message
- SOAP defines set of encoding rules, based on XML Schema
  - SOAP-ENV:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/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

SOAP-based messaging

- SOAP is fundamentally a stateless, one-way message exchange paradigm
  - …but applications can create more complex interaction patterns
    - Request/response, request/multiple responses
  - Binding to HTTP (synchronous protocol) makes RPC-style "natural"
    - One-way exchange will use simple acknowledgement as HTTP response
- SOAP-based RPC
  - Invocation is modelled as a struct of in/inout parameters
    - <doCheck>
      <product> … </product>
      <quantity> … </quantity>
      </doCheck>
  - Response is modelled as a struct as well
    - <doCheckResponse> … </doCheckResponse>
  - All data is passed by-value
- Non-RPC SOAP messaging
  - Document-centric
  - No rules for operation names, encoding of parameters
- More on this when we discuss WSDL …

More SOAP

- 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
    - Example: separate authentication/authorization from service implementation
    - Intermediary does not need to understand message body
  - A message can required that a specific header is processed by the NEXT node (either intermediary or recipient)
- 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, …



WSDL

How To Define New Web Services

- Well, it's nice to refer to web services that others have defined and agreed on (= tModels)
- ...but what if I want to publish my own web services
  - e.g. simple functions useful for others
  - or complete business processes that I offer
- ...without creating a standard!
- We need a language to specify services and describe how others can bind to them!

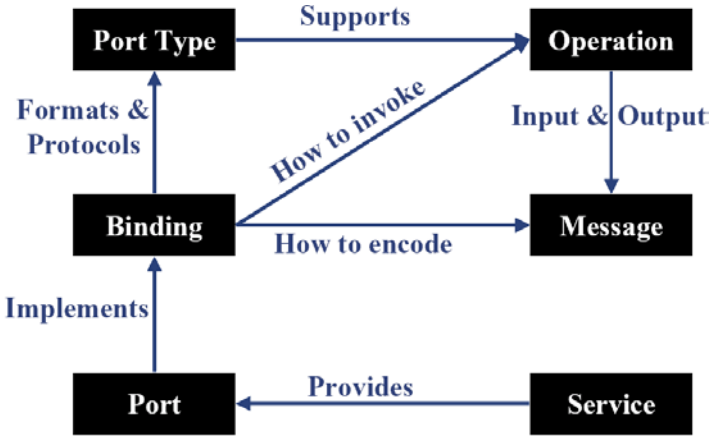
WSDL

- Web Services Description Language
  - Provides all information necessary to programmatically access a service
  - XML-based language to specify a web service
    - Name of service
    - URL of service manager
    - Available methods
    - In/Out parameters of methods
- Tools generate code out of the service specification
  - IBM WSTK: wsdlgen
  - systinet WASP: WSDLCompiler
  - Apache Axis: wsdl2java
  - Others
- WSDL V1.1 specification
  - http://www.w3.org/TR/wsdl

Ingredients of WSDL

- Types: Definitions of data types needed
- Message: Abstract definition of data exchanged
- Operation: Abstract actions supported by the service
- Port Type: Set of operations supported by an „end point"
- Binding: Concrete protocol and data format used to implement a port type
- Port: Single individual „end point" identified by a network address supporting a particular binding
- Service: Collection of related „end points"

How the Ingredients Relate



WSDL Document Structure

```
<definitions name="nmtoken"? targetNamespace="uri"?>
  <import namespace="uri" location="uri"/>*
  <documentation…/>?
  <types>?
  <message name="nmtoken">*
  <portType name="nmtoken">*
  <binding name="nmtoken" type="qname">*
  <service name="nmtoken">*
  <-- extensibility element -->*
</definitions>
```

*= 0 or more
?= 0 or 1

Port Types

```
<portType name="nmtoken">
  <operation name="nmtoken" parameterOrder="nmtokens">*
    <input name="nmtoken"? message="qname"/>?
    <output name="nmtoken"? message="qname"/>?
    <fault name="nmtoken" message="qname"/>*
  </operation>
</portType>
```

- Port type is a set of abstract operations and abstract messages involved
- Four kinds of operations are supported
  - One-way
  - request-response
  - solicit-response
  - Notification
- parameterOrder is list of message parts and may be used to specify signature of an operation

Transmission Primitives

- Request-response
  - Both, input and output messages must be specified
  - Concrete binding must be consulted to determine how messages are sent
    - E.g. single HTTP request/response, two HTTP requests,...
- One-way
  - No output and no fault messages are specified
- Notification
  - No input and no fault messages are specified
  - Like a one-way push from the service provider
    - Where to send to? Outside scope of WSDL
      - Information could be provided through another (subscribe) operation
- Solicit-response
  - Both output/fault and input messages have to be specified
  - Push from service provider, expecting an input (response) from service requestor in reply
    - Where to send to …

Messages

```
<message name="nmtoken">*
    <part name="nmtoken"?
     element="qname"
     type="qname"?/>*
</message>
```

- Message consists of one or more logical parts
- Parts are typed
- May be used for
  - specifying each parameter of an RPC
  - Logically structuring complex messages
- Typically, message is „abstract"
  - Concrete format specified via bindings

Example

```
<?xml version="1.0"?>
<definitions name="StockQuote"
    xmlns:tns="http://myservice.com/stockquote.wsdl"
    xmlns:xsd="http:// myservice.com/stockquote.xsd"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
 <message name="GetLastTradePriceRequest">
    <part name="tickerSymbol" element="xsd:string"/>
    <part name="time" element="xsd:timeInstant"/>
 </message>
 <message name="GetLastTradePriceResponse">
    <part name="result" type="xsd:float"/>
 </message>
 <portType name="StockQuotePortType">
    <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceRequest"/>
          <output message="tns:GetLastTradePriceResponse"/>
    </operation>
 </portType> ...
```

Types

<definitions…>
<**types**>
<xsd:schema…/>*
</types>
</definitions>

- Default type system is XSD (XML Schema)
- Special extensibility element foreseen to refer to other type system
- Type clause used to define types in messages
  - Wire format may or may not be XML
- Example

```
<definitions name="StockQuote" …>
    <types>
      <xsd:schema …>
          <xsd:complexType name="registrationRequest">
              <xsd:sequence>
…
```


Binding

- portType, message, 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
  - portType can have one or more bindings
- Protocol-specific aspects are provided using binding extensions

```
<binding name="nmtoken" type="qname">
    <-- extensibility element (1) -->*
    <operation name="nmtoken">*
      <-- extensibility element (2) -->*
      <input name="nmtoken"?>?
          <-- extensibility element (3) -->*
      </input>
      <output name="nmtoken"?>?
          <-- extensibility element (4) -->*
      </output>
      <fault name="nmtoken">*
          <-- extensibility element (5) -->*
      </fault>
    </operation>
</binding>
```

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

Example – SOAP Binding

```
<binding name="StockQuoteSoapBinding"
    type="tns:StockQuotePortType">
    <soap:binding
        style="rpc"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="GetLastTradePrice">
        <soap:operation
            soapAction="http://myservice.com/GetLastTradePrice"/>
        <input>
            <soap:body use="encoded"
                namespace="http://myservice.com/stockquote"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </input>
        <output>
            <soap:body use="encoded"
                namespace="http:// myservice.com/stockquote"
                encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
        </output> ...
```


Port and Service

- Port
  - Specifies the network address of the endpoint hosting the web service
- Service
  - Contains a set of related port elements
    - Group ports related to the same service interface (portType) but expressed by different protocols (bindings)
    - Group related but different port types together
- Example

```
<service name="StockQuoteService">
<documentation>My first service</documentation>
<port name="StockQuotePort“
    binding="tns:StockQuoteSoapBinding">
    <soap:address
      location="http://myservice.com/stockquote"/>
</port>
</service>
```

Modularizing Service Definitions

<definitions…>
  <**import** namespace="uri" location="uri"/>*
</definitions>

- Can be used to factor out any kind of definitions
  - Types, Messages, PortTypes, Bindings,…

  or any combination of these
- Example:
  - Import PortType and specify Binding
  - Import Binding and specify Service
- Helps writing clearer definitions
- Enables reuse
- Resulting documents are easier to maintain

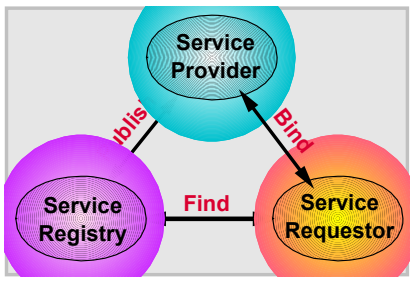
UDDI

How To Find Services

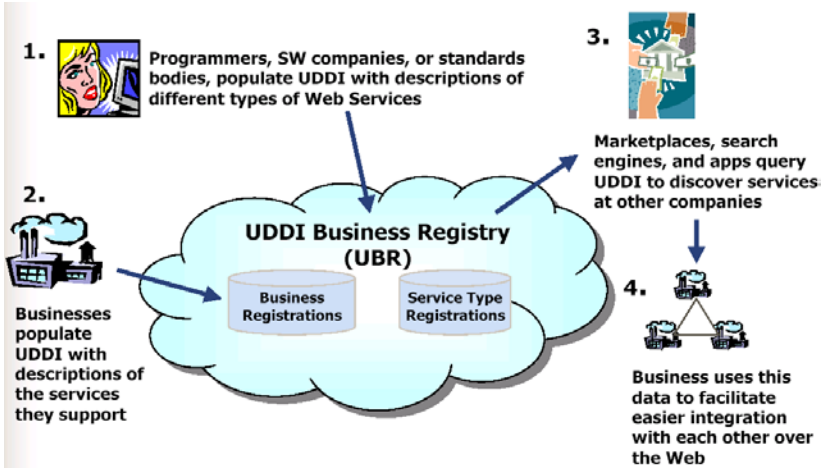
- What businesses offer services I need?
- What do I have to do to interface with these services?
- Who currently offers services I am configured to use?
- ⇨ We need a globally available directory
  - ...to catalogue services based on publish requests of service providers
  - ...to maintain taxonomy(ies) to support searching for appropriate services in business terms
  - ...to specify technical binding information to actually communicate with the selected service

Universal Description Discovery and Integration (UDDI)

- UDDI is three things:
  - Set of schemas for describing businesses and their services
  - Global registry of businesses and their services
    - can include a pointer to WSDL
  - SOAP API for accessing a UDDI registry
- Serves as the directory of Web services
  - Allows searching "by what" and "by how" instead of just "by name"
- UDDI initiative
  - Involves more than 300 companies
  - http://www.uddi.org



How UDDI Works



Registry Data

- Businesses register public information about themselves
  - White pages
    - Who am I?
  - Yellow pages
    - What do I offer?
  - Green pages
    - How to do business with me
- Standards bodies, Programmers, Businesses register information about their Service Types („tModels")
  - Service Type Registrations

White Pages

- Business Name
- Text Description
  - list of multi-language text strings
- Contact info
  - names, phone numbers, fax numbers, web sites…
- Known Identifiers
  - list of identifiers that a business may be known by
    - DUNS, Thomas, other

Business Entity

Yellow Pages

- Business categories
  - 3 standard taxonomies in V1
    - Industry: NAICS (Industry codes – US Govt.)
    - Product/Services: UN/SPSC (ECMA)
    - Location: Geographical taxonomy
  - Implemented as name-value pairs to allow any valid taxonomy identifier to be attached to the business white page

Business Service

Green Pages

- New set of information businesses use to describe how to "do e-commerce" with them
  - Nested model
    - Business processes
    - Service descriptions
    - Binding information
- Programming/platform/implementation agnostic
- Services can also be categorized

Binding Template

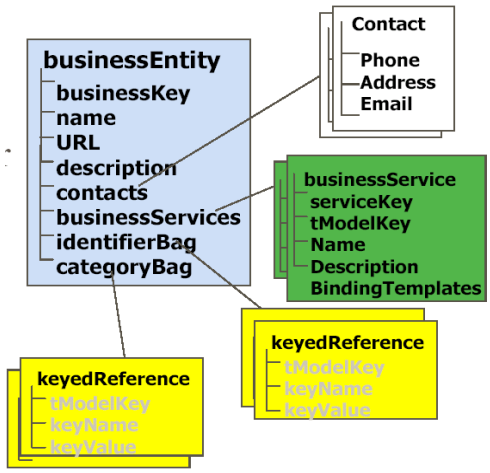
Service Type Registration

- Pointer to the namespace where service type is described
  - What programmers read to understand how to use the service
    - Identifier for who published the service
  - Identifier for the service type registration
    - called a tModelKey
    - Used as a signature by web sites that implement those services

tModel

Business Registration

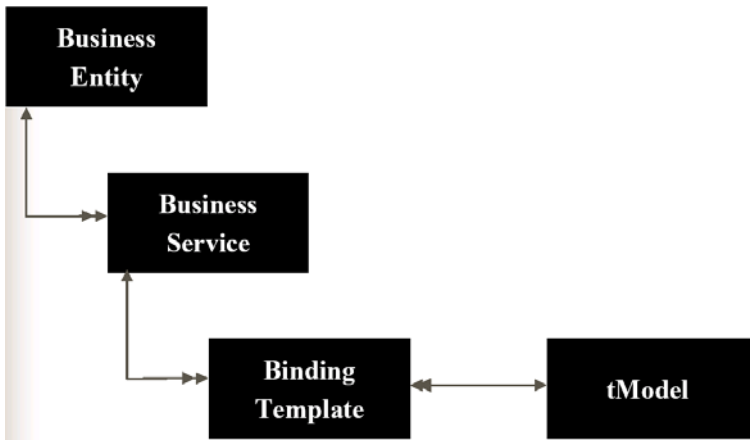
- XML document
- Created by end-user company (or on their behalf)
- Can have multiple service listings
- Can have multiple taxonomy listings



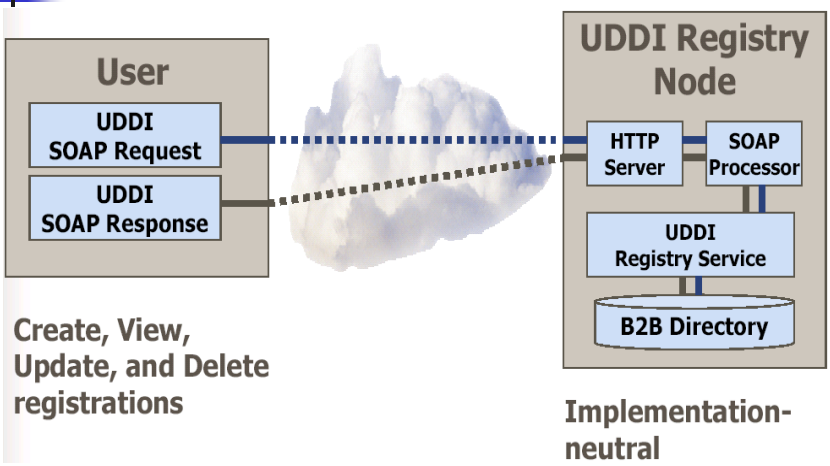
Registry Operation

- Peer nodes (websites)
  - IBM
  - Ariba
  - other
- Companies register with any node
- Registrations replicated on a daily basis
- Complete set of "registered" records available at all nodes
- Common set of SOAP APIs supported by all nodes
- Browser-based interfaces are provided as well
- Compliance enforced by business contract

Major UDDI Data Structures



UDDI and SOAP



Registry APIs (SOAP messages)

- 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

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

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
- When registering a tModel it gets a globally unique identifier: tModelKey
- tModelKey is like a „fingerprint" for the concept, idea,...
- For example, tModelKeys describe the semantics of
  - 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,...

Structure of a tModel

```
<element name = "tModel">
  <type content = "elementOnly">
    <group order = "seq">
      <element ref = "name"/>
      <element ref = "description" minOccurs = "0" maxOccurs = "*"/>
      <element ref = "overviewDoc" minOccurs = "0" maxOccurs = "1"/>
      <element ref = "identifierBag" minOccurs = "0" maxOccurs = "1"/>
      <element ref = "categoryBag" minOccurs = "0" maxOccurs = "1"/>
    </group>
    <attribute name = "tModelKey" minOccurs = "1" type = "string"/>
    <attribute name = "operator" type = "string"/>
    <attribute name = "authorizedName" type = "string"/>
  </type>
</element>
```

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,...?