Web Services Foundations

Worflows und Web Services
Kapitel 3

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
    - provide a common definition of a web service
    - define its place within a larger Web services framework to guide the community
  - public working draft
- Definition in the scope of this WG still evolving
  - "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
  - "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."
    August 2003

Web Services Evolution – Organizing Software Granules

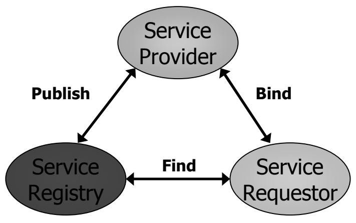
- Subroutines, Functions
  - Centered around functional decomposition
  - Allows a system to be modularized, i.e. subdivided
  - Results into concept of APIs
- Objects
  - Centered around combining functions and data into encapsulated units
  - Enables concepts of classes, inheritance, polymorphism
  - 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

Software Aspect: Web Services

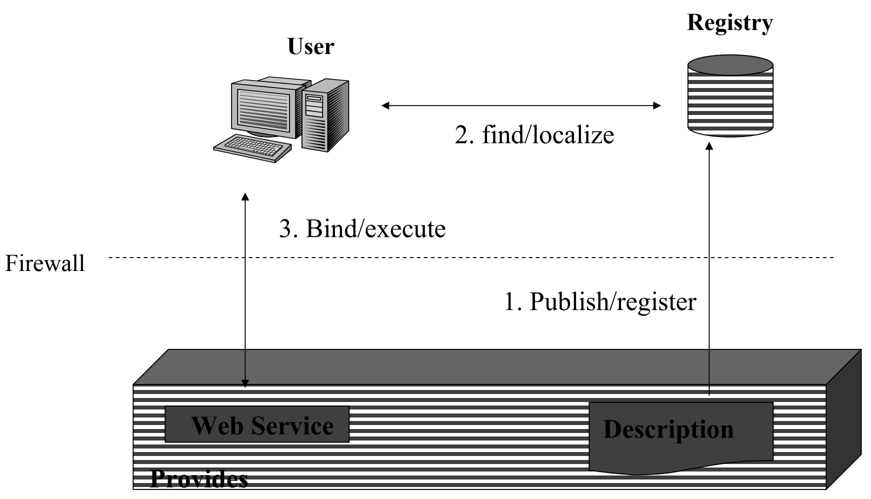
- A web service is a piece of software made available on the Web
- An architecture is service based iff it focuses on
  - formats and protocols for communication between services
    - E.g. RosettaNet, OBI,...
- An architecture is service oriented iff it focuses on
  - How to support the dynamic discovery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

Service-Oriented Architecture (SOA)

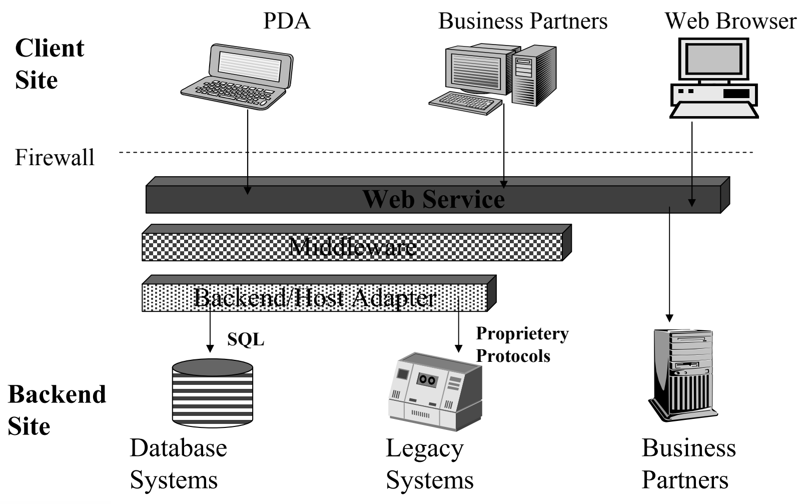
- Service Requestor
  - Finds required services via Service Broker
  - Binds to services via Service Provider
- Service Provider
  - Provides e-business services
  - Publishes availability of these services through a registry
- Service Registry
  - Provides support for publishing and locating services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



Web Service Model



Web Service System Architecture



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

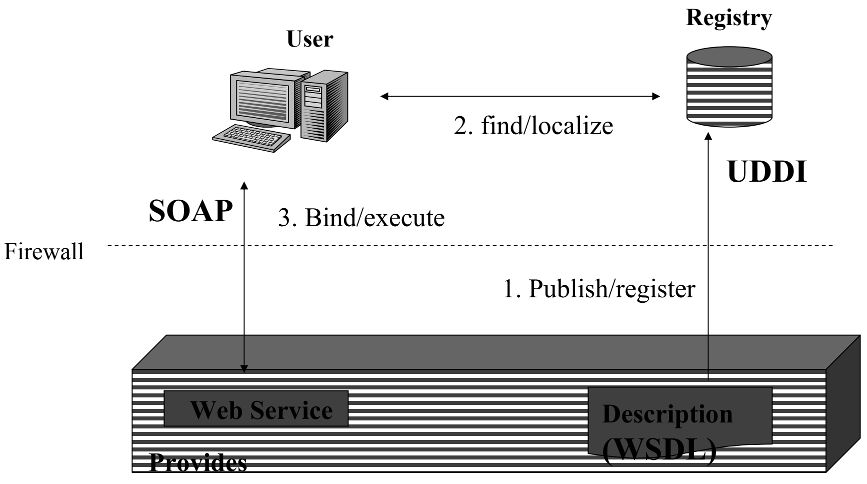
Class Hierarchies vs. Taxonomies

- 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

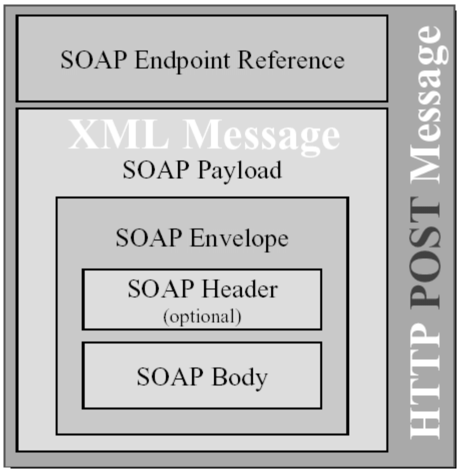
- Existing technologies like CORBA/IIOP, DCOM, or EJB are strong for server-to-server communication
  - ...but rely on closely administered environment
    - E.g. very cumbersome to allow two random computers to call each other out-of-the-box
- Weaknesses
  - Clients scattered across the Internet
  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

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
- Based on TCP, connection oriented
- Can easily reach across firewalls

SOAP over HTTP

- HTTP limitations
  - payload is only a single parameter of type "string"
  - we need a rich type system for exchanging parameters
    - XML Schema
- SOAP: Simple Object Access Protocol
  - Can use existing technologies
    - HTTP & XML ("XML as HTTP payload")
      - HTTP as RPC transport
      - XML as RPC encoding scheme
  - No special SOAP API, no special SOAP ORB
- … but SOAP is much more than SOAP/HTTP!

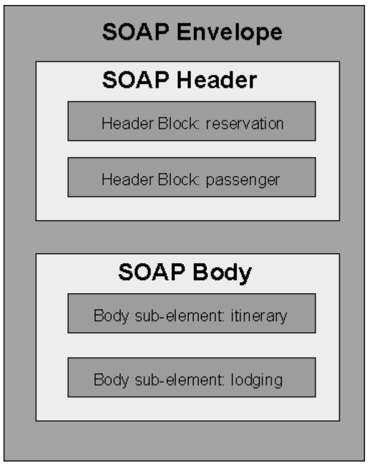


What Does SOAP Provide?

- SOAP Message as unit of communication
  - Envelope, body, headers
  - Flexible mechanism for data representation
- Stateless, one-way message exchange
  - more complex interaction patterns (e.g., request/response) must be defined on top
    - exploiting features of underlying protocols
    - adding specific information to messages
- RPC mechanism
  - Calls and responses represented as SOAP messages
- Document-centric approach
  - Alternative to fine-grained RPC interfaces
- Extensibility mechanisms
  - XML, schemas, namespaces
  - SOAP headers for protocol extensions
- SOAP Fault mechanism for error handling
- Binding framework and bindings to protocols (HTTP, …)

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 Terminology

- node
  - Processing logic necessary to transmit, receive, process and/or relay a SOAP message, according to the set of conventions defined by this recommendation. Responsible for enforcing the rules that govern the exchange of SOAP messages. Accesses the services provided by the underlying protocols through one or more SOAP bindings.
- role
  - A SOAP receiver's expected function in processing a message. A SOAP receiver can act in multiple roles.
- binding
  - Formal set of rules for carrying a SOAP message within or on top of another protocol (underlying protocol) for the purpose of exchange.
- feature
  - Extension of the SOAP messaging framework. Examples of features include "reliability", "security", "correlation", "routing", and "Message Exchange Patterns" (MEPs).
- message exchange pattern (MEP)
  - Template for the exchange of SOAP messages between SOAP nodes.

SOAP Terminology (cont.)

- 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 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="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

- 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
  - 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

```
<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

Message Exchange Patterns

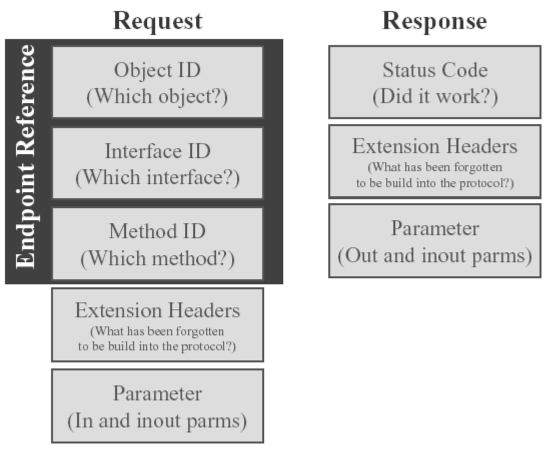
- Template that establishes a pattern for the exchange of messages between SOAP nodes
  - Example: request-response MEP specified in SOAP 1.2 Part 2
- An MEP must
  - provide a URI to name the MEP
  - describe the life cycle of a message exchange conforming to the pattern
  - describe the temporal/causal relationships, if any, of multiple messages exchanged in conformance with the pattern (e.g. responses follow requests and are sent to the originator of the request)
  - describe the normal and abnormal termination of a message exchange conforming to the pattern
  - any requirements to generate additional messages (such as responses to requests in a request/response MEP)
  - rules for the delivery or other disposition of SOAP faults generated during the operation of the MEP
- Protocol Bindings
  - can claim support for specific MEPs
    - taking advantage of underlying protocol, or
    - build "on top" using binding-specific extensions

SOAP-based RPCs

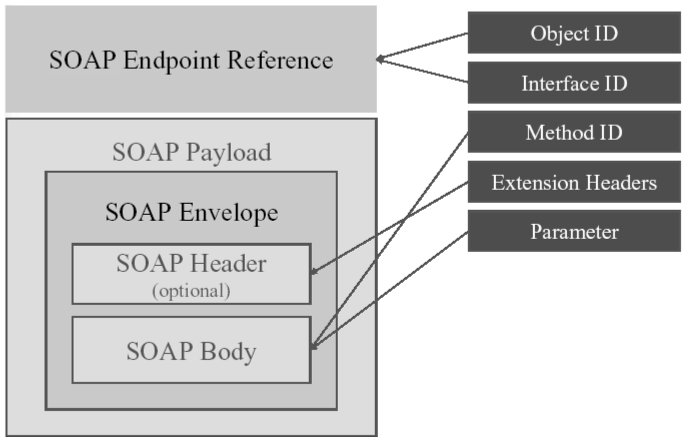
- SOAP is fundamentally a stateless, one-way message exchange paradigm
  - …but applications can create more complex interaction patterns
    - Request/response, request/multiple responses
- SOAP-based RPC
  - Employs request/response MEP
  - Invocation is modelled as a struct of in/inout parameters

    -
      ```
      <doCheck>
                  <product> … </product>
                  <quantity> … </quantity>
      </doCheck>
      ```
  - Response is modelled as a struct as well
    - <doCheckResponse> … </doCheckResponse>
  - All data is passed by-value
  - 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

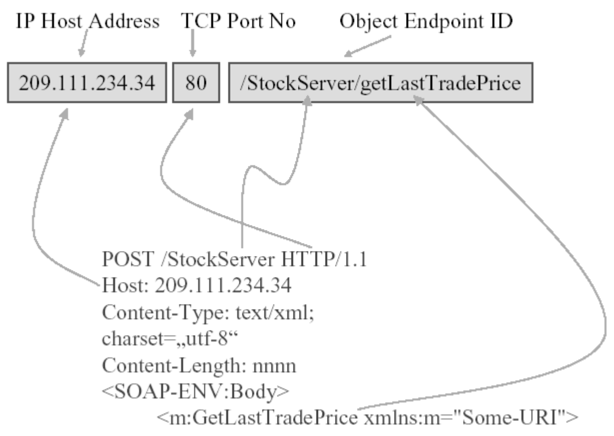
ORPC Request/Response



RPC via SOAP



SOAP/HTTP Endpoint Reference



```
POST /StockServer HTTP/1.1
Host: 209.111.234.34
Content-Type: text/xml;
charset=„utf-8“
Content-Length: nnnn
<SOAP-ENV:Body>
        <m:GetLastTradePrice xmlns:m="Some-URI">
```

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
  - industry specification published by BEA, IBM, and Microsoft

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: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

Processing Model

- Each message has a globally unique ID, message sequence id, incremental id within sequence
  - specification defines SOAP header structure for this information
- Receiver must correctly acknowledge successful receipt of messages
  - references the corresponding message id
- If sender does not receive acknowledgement it retries submission of message until maximum number of retries is reached
  - corresponding fault must be thrown to application

Ordering Model

- Messages within a group have a sequence number
  - increased by "1" for every message
- Receiver can deliver a message to application layer if all messages received have consecutive sequence number
  - messages are to be delivered ordered by sequence number
  - if a message is missing, only the ones that appear in the sequence order before the missing one can be delivered
- If the sender receives an acknowledgement for the final message but acks for earlier messages are missing, submission of missing messages must be retried

Example

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
      xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/rm"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <S:Header>
      <wsa:MessageID>
            http://Business456.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
      </wsa:MessageID>
      <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
      <wsa:ReplyTo>
            <wsa:Address>http://Business456.com/serviceA/789</wsa:Address>
      </wsa:ReplyTo>
      <wsrm:Sequence>
            <wsu:Identifier>http://Business456.com/RM/ABC</wsu:Identifier>
            <wsrm:MessageNumber>3</wsrm:MessageNumber>
            <wsrm:LastMessage/>
      </wsrm:Sequence>
  </S:Header>
  <S:Body>
      <!-- Some Application Data -->
  </S:Body>
</S:Envelope>
```

This is the 3rd message in the sequence

Example (cont.)

```
<?xml version="1.0" encoding="UTF-8"?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
      xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
      xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/rm"
      xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
  <S:Header>
      <wsa:MessageID>
            http://fabrikam123.com/guid/0baaf88d-483b-4ecf-a6d8-a7c2eb546817
      </wsa:MessageID>
      <wsa:To>http://Business456.com/serviceA/789</wsa:To>
      <wsa:ReplyTo>
            <wsa:Address>http://fabrikam123.com/serviceB/123</wsa:Address>
      </wsa:ReplyTo>
      <wsrm:SequenceAcknowledgment>
            <wsu:Identifier>http://Business456.com/RM/ABC</wsu:Identifier>
            <wsrm:AcknowledgmentRange Upper="1" Lower="1"/>
            <wsrm:AcknowledgmentRange Upper="3" Lower="3"/>
      </wsrm:SequenceAcknowledgment>
  </S:Header>
  <S:Body/>
</S:Envelope>
```

Message 2 was not received
Sender has to retry

WSDL

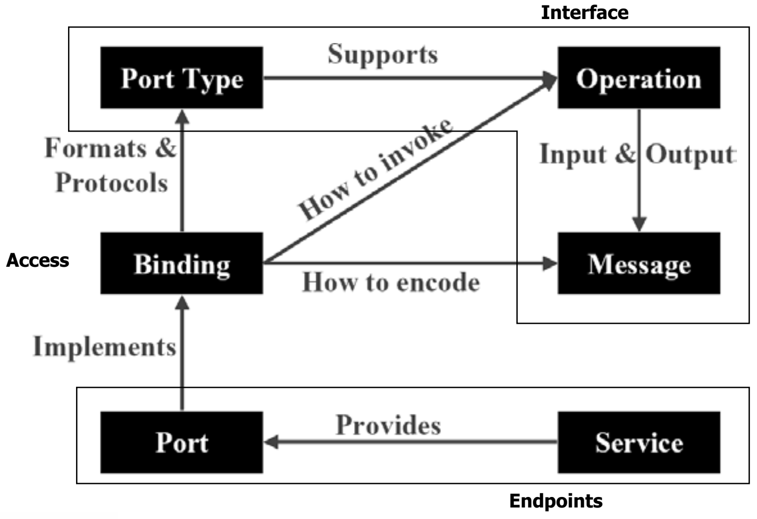
WSDL

- Web Services Description Language
  - Provides all information necessary to programmatically access a service
    - documentation for distributed systems
    - recipe for automating the details involved in applications communication
  - XML-based language to specify a web service
    - Name of service
    - URL of service manager
    - Available methods
    - In/Out parameters of methods
    - Bindings to specific protocols and data formats
- 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
  - Next version of WSDL is a w3c working draft

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 (in this order)
  - 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>
```

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

SOAP Binding - Details

- <soap:binding>
  - transport: HTTP, SMTP, FTP, …
  - style: default style for operations
- <soap:operation>
  - soapAction: value of SOAPAction HTTP header (SOAP over HTTP only!)
  - style: actual style for the operation (rpc | document)
    - rpc: message parts are rpc parameters
      - wrapped in parameter, operation elements, see "SOAP-based RPCs"
    - document: message parts are documents (this is also the default)
- <soap:body>
  - parts: message parts accuring in the body, in the order specified
    - some parts may become headers …
  - use: how the concrete message is produced
    - literal: use the concrete schema defined for message parts
    - encoded: use the abstract types defined for message parts, apply encoding style
- <soap:header> *
  - message, parts: which parts of which messages are included as headers

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>
```

implemented binding

binding-specific address of the port

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

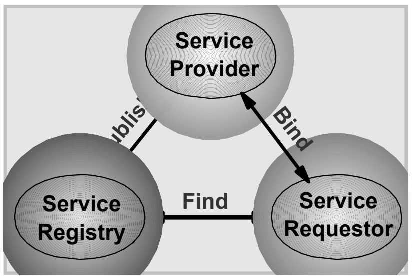
UDDI

How To Find Services

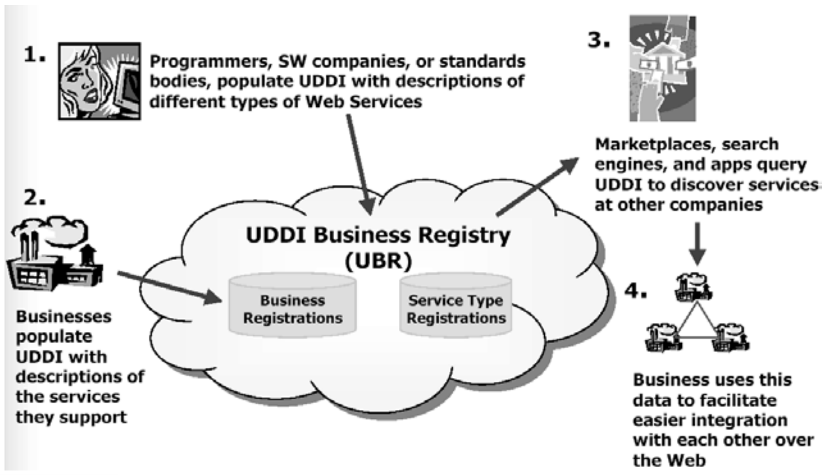
- What businesses offer services I need?
- What do I have to do to interface with these services?
- Who currently offers services I am configured to use?
- ➪ We need a globally available directory
  - ...to catalogue services based on publish requests of service providers
  - ...to maintain taxonomy(ies) 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

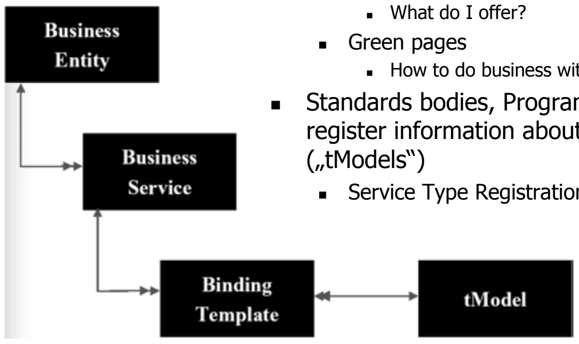


How UDDI Works


1.
Programmers, SW companies, or standards bodies, populate UDDI with descriptions of different types of Web Services
2.
Businesses populate UDDI with descriptions of the services they support
UDDI Business Registry (UBR)
Business Registrations
Service Type Registrations
3.
Marketplaces, search engines, and apps query UDDI to discover services at other companies
4.
Business uses this data to facilitate easier integration with each other over the Web

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
  - standard taxonomies
    - 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

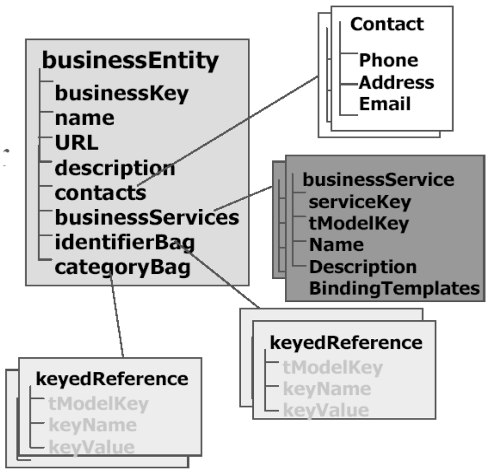
Service Type Registration

- 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

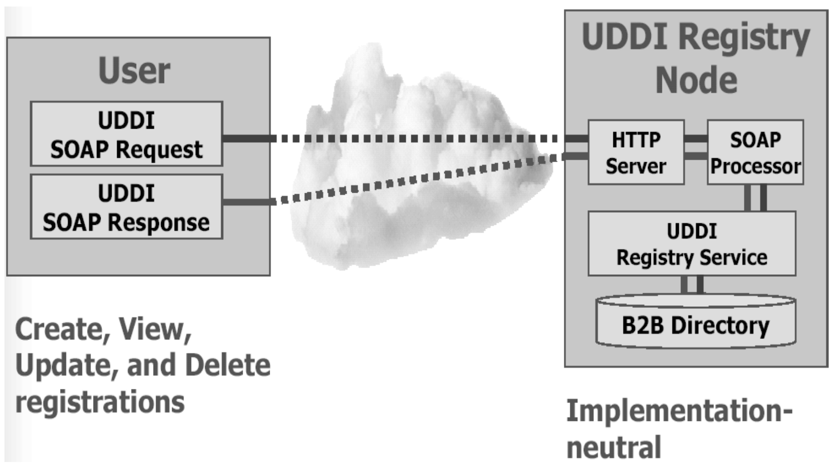
- XML document
- Created by end-user company (or on their behalf)
- Can have multiple service listings
- Can have multiple taxonomy listings



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

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,...?
  - 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" />
```

Discovering Web Services – Without UDDI

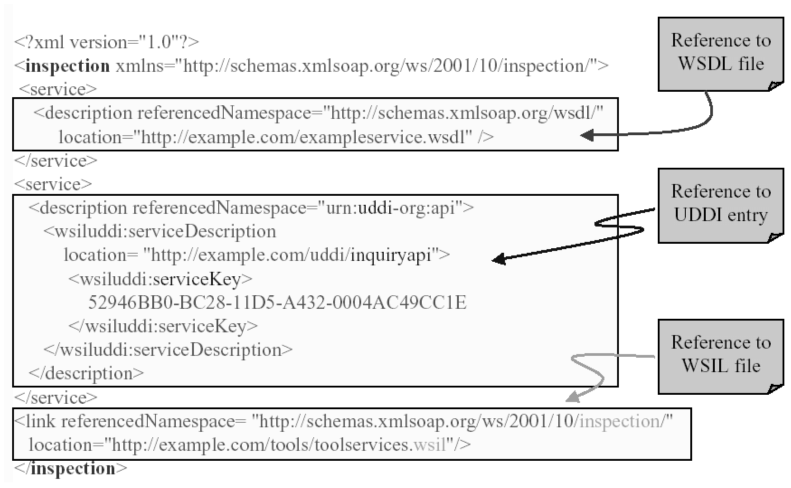
- Sometimes you don‘t want to register (yet) a Web Service in UDDI
  - It may not be of public interest
  - It may not be ready for production
  - ...
- Web Services Inspection Language (WSIL)
  - Language to discover Web Services at Web sites
  - Proposed by IBM and Microsoft (11/2001)
  - Supported 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 informations 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



```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
 <service>
   <description referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
       location="http://example.com/exampleservice.wsdl" />
</service>
<service>
  <description referencedNamespace="urn:uddi-org:api">
    <wsiluddi:serviceDescription
        location= "http://example.com/uddi/inquiryapi">
        <wsiluddi:serviceKey>
           52946BB0-BC28-11D5-A432-0004AC49CC1E
        </wsiluddi:serviceKey>
    </wsiluddi:serviceDescription>
  </description>
</service>
<link referencedNamespace= "http://schemas.xmlsoap.org/ws/2001/10/inspection/"
  location="http://example.com/tools/toolservices.wsil"/>
</inspection>
```

Referencing WSDL Elements

```
<?xml version="1.0"?>
<inspection xmlns="http://schemas.xmlsoap.org/ws/2001/10/inspection/">
<service>
<name xml:lang="en-US">StockQuoteService</name>
<description referencedNamespace="http://schemas.xmlsoap.org/wsdl/">
<wsilwsdl:reference
      endpointPresent="true"
      location="http://localhost:8080/webservices/wsdl/stockquote/sqs.wsdl">
      <wsilwsdl:referencedService
                xmlns:tns="http://www.getquote.com/StockQuoteService">
                tns:StockQuoteService
      </wsilwsdl:referencedService>
      <wsilwsdl:implementedBinding
                xmlns:interface="http://www.getquote.com/StockQuoteService-interface">
                interface:StockQuoteServiceBinding
      </wsilwsdl:implementedBinding>
</wsilwsdl:reference>
</description>
</service> .
</inspection>
```

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" – port type, operation, message
    - binding to specific protocol (e.g., SOAP)
      - protocol extensions
    - "endpoint" – port, 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