T-110.5140 SOAP and UDDI

Tancred Lindholm, Sasu Tarkoma and Pekka Nikander

Aalto University

Lecture outline

- SOAP
  - Document style vs. RPC style SOAP
  - SOAP intermediaries
  - Data encoding in SOAP
- UDDI
  - White, Yellow and Green pages
  - UDDI API

SOAP I

- W3C XML Protocol Working Group is specifying SOAP, part of the Web Services Activity
  - SOAP 1.2 currently a W3C Recommendation
- An application of the XML specification
  - XML Infoset
- Relies on XML Schema, XML Namespaces
- Platform independent
- Provides a standard way to structure XML Messages

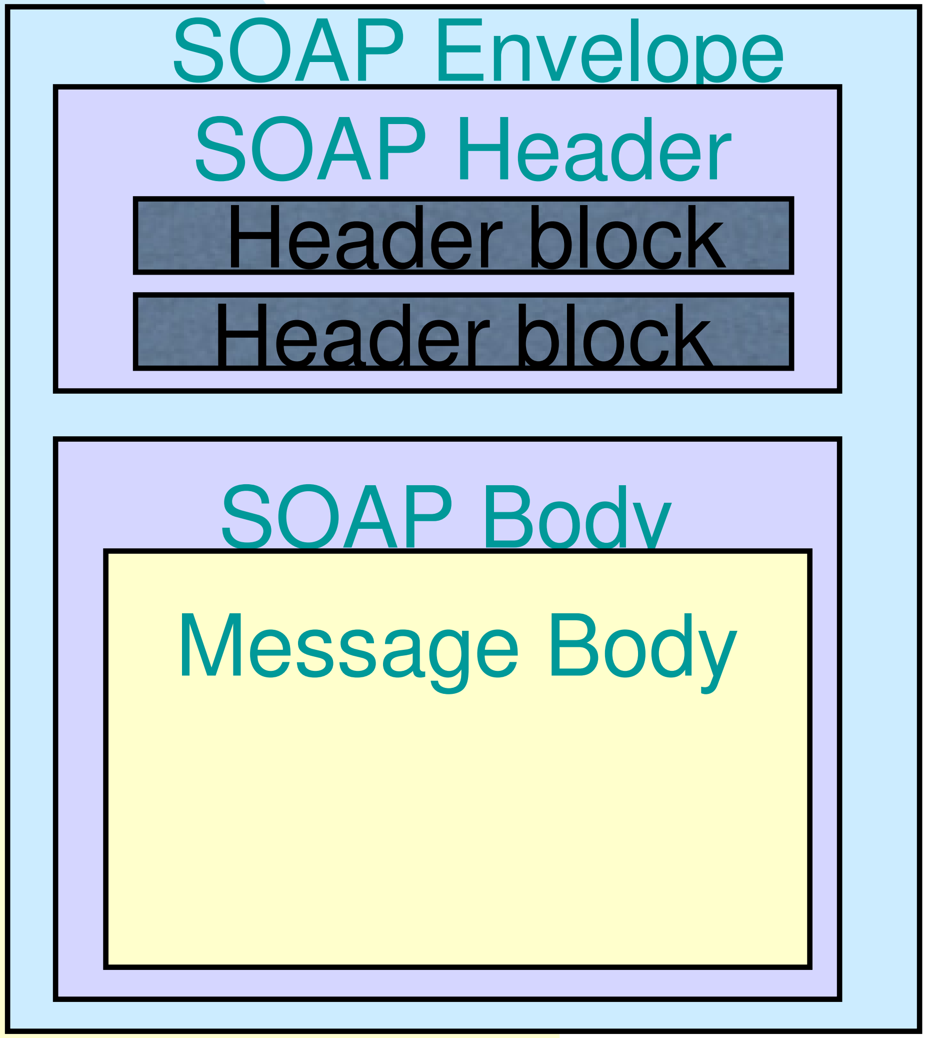
SOAP II

- Fundamentally stateless one-way message exchange paradigm
  - More complex interactions may be implemented
- Exchange of structured and typed information
  - Between peers in decentralized fashion
  - Using different mediums: HTTP, Email, ..
- Request-reply and one-way communication are supported
- Note that XML infoset is an abstract specification
  - On-the-wire representation does not have to be XML 1.0!
- Specifications
  - SOAP Version 1.2 Part 0: Primer
  - SOAP Version 1.2 Part 1: Messaging Framework
  - SOAP Version 1.2 Part 2: Adjuncts
  - SOAP Version 1.2 Specification Assertions and Test Collection

It is necessary to define:

- The type of information to be exchanged
- How to express the information as XML (according to the Infoset)
- How to send the information
- SOAP defines (adjuncts part) these using:
  - Data model
    - Application-defined data structures and values as a directed, edge-labeled graph
  - SOAP encoding
    - Rules for encoding instances of data from SOAP data model to XML
  - One-way and request-reply (RPC) msg exchange
  - Binding framework in order to support concrete messaging protocols and custom on-the-wire representation

SOAP Message Structure



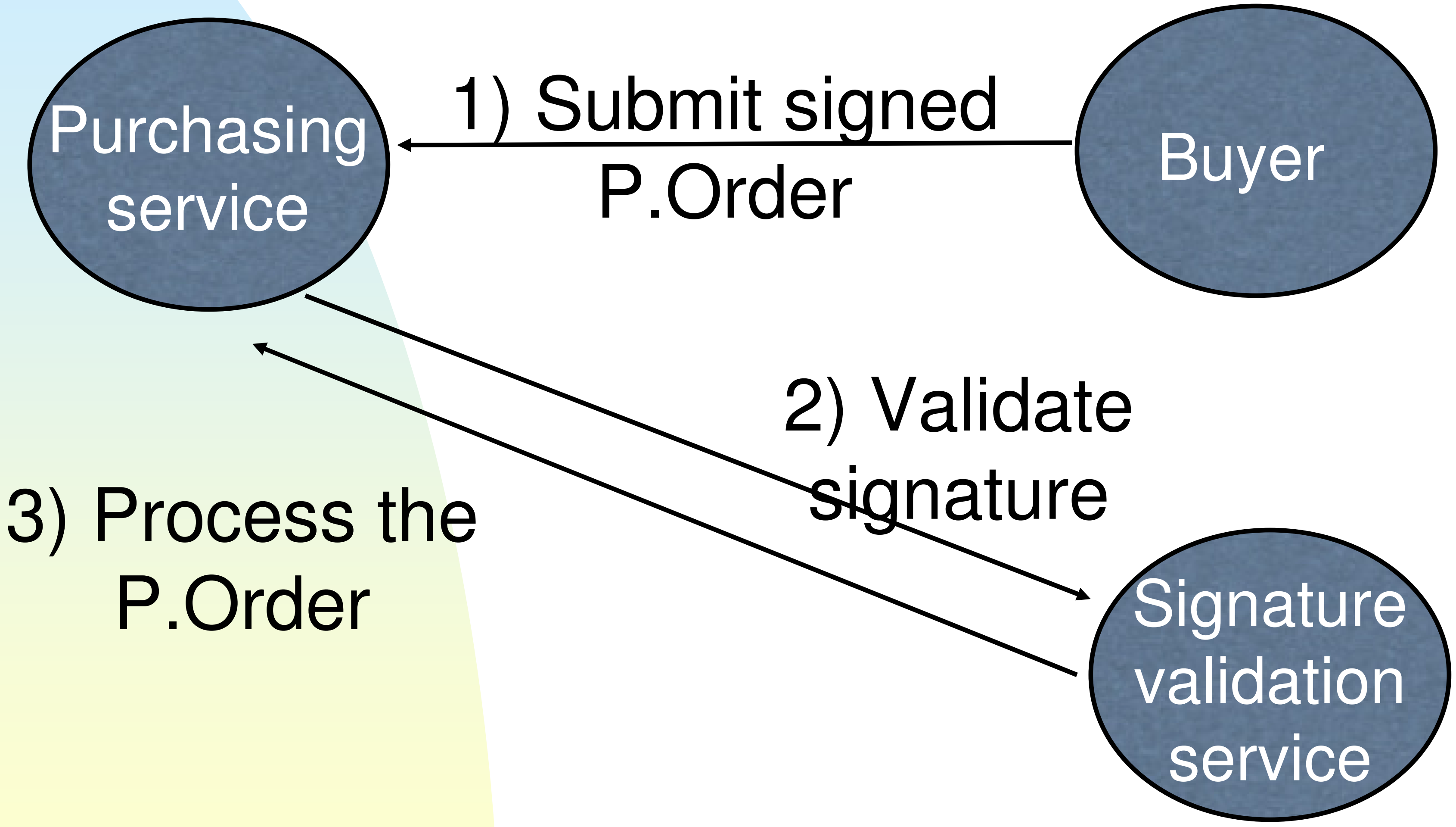
Optional header contains blocks of information regarding how to process the message:
- Routing and delivery settings
- Authentication/authorization assertions
- Transaction contexts

- Body is a mandatory element and contains the actual message to be delivered and processed (and fault information)

SOAP Message Exchange Model

- SOAP intermediary, or actor, sits between a service consumer and provider and adds value or functionality to the transaction
- The set of intermediaries that the message travels through is called the message path
- No standard way of expressing the message path
- “targeting:” SOAP has a mechanism for identifying which parts of the SOAP message are intented for processing by specific actors in its message path
    - Only for header blocks: targeted to a specific actor on its message path by using “actor” attribute

Intermediary example



Intermediaries

- SOAP Version 1.2 describes two intermediaries
- A Forwarding Intermediary
  - forwards SOAP messages
  - ”routing” block
  - May not modify content
- An Active Intermediary
  - Additional processing on an incoming SOAP message
  - Headers, message excange pattern
  - May modify content in the message
  - encryption, blinding, new header block, timestamping, annotation, ..

The ”role” attribute

- Processing of header block and the body depend on the role(s) assumed by the SOAP node for the message
  - SOAP defines optional env:role attribute that
  - may be present in a header block (a URI)
  - identifies the role played by the intended target of the block
- A SOAP node is required to process the block if it assumes the role identified by the value of the URI
- Three standardized roles:
  - None
    - no SOAP node should process the block
  - Next
    - next node must process block
  - ultimateReceiver
    - implicit if role not specified
- It is up to the node to know its roles, not part of the specification

Roles continued

- Note that env:Body is always targeted to the ultimate receiver and it must be processed
- Mandatory header blocks (mustUnderstand=true) must be processed if the node has the role identified in the mandatory block

SOAP Header II

- The SOAP rules require that processed blocks are removed from the outbound message
- Unprocessed header blocks targeted at a role played by a SOAP intermediary are also removed
- The “relay” attribute may be used to preserve the unprocessed header blocks
  - SOAP 1.2 feature

Header example

```
<m:reservation xmlns:m="http://travelcompany.example.org/reservation"
         env:role="http://www.w3.org/2003/05/soap-envelope/role/next"
          env:mustUnderstand="true">
```

The next SOAP node must process this header block. The block is removed (even if not processed), but may be reinserted. Preserved when ”relay” is used.

SOAP RPC I

- SOAP may be used for both request-reply and one-shot messaging
- Ultimate SOAP receiver is the target of the RPC procedure
- RPC information is carried in the env:Body element and modelled as a struct / array
- Serialization according to the SOAP encoding rules
    - They are optional and there may be several encoding rules for data types

SOAP RPC II

- To make an RPC call the following information is needed:
  - The address of the target SOAP node (ultimate receiver)
  - The procedure name
  - The identities and values of any arguments, output parameters and return value
  - A clear separation of the arguments, which is the target and what is additional information
  - The message exchange pattern
  - Optional data carried in the header blocks
- Service and procedure specification is stored in a WSDL file

SOAP’s Data Encoding

- Method of serializing the data intended for packaging
- Rules outline how basic application data types are to be mapped and encoded into XML
- A simple type system that is a generalization of the common features found in type systems in programming languages, databases, etc.
- SOAP encoding and SOAP RPC representation are optional and not always useful
  - Encoding not needed if the data is already in XML
  - RPC representation has some restrictions on data models and encodings for RPC calls

Encoding Styles

- SOAP RPC encoding (rpc/encoded)
  - <soap:Body> contains an element with the name of the method or remote procedure being invoked
  - This element in turn contains an element for each parameter of the procedure
  - SOAP stack handles the complexity
  - Section 5 of the SOAP 1.1 spec, mapping into XML 1.0
  - Developed before schema / WSDL
- SOAP RPC Representation literal encoding (rpc/literal)
  - Suitable for XML data
  - Schema for every parameter type but not for the whole body
- SOAP document-style literal (document/literal)
  - There are no SOAP formatting rules for what the <soap:Body> contains
  - The developer handles everything (using schemas)
  - Easier for the system, easy to validate body
  - Web Service Interoperability (WS-I): Only document/literal allowed!

Purchase order in document/literal-style SOAP

```
<s:Envelope
      xmlns:s=http://www.w3.org/2001/06/soap-envelope>
      <s:Header>
         <m:transaction xmlns:m="soap-transaction"
                  s:mustUnderstand="true">
            <transactionID>1234</transactionID>
          </m:transaction>
       </s:Header>
       <s:Body>
          <n:purchaseOrder xmlns:n="urn:OrderService">
            <from><person>Christopher Robin</person></from>
            <to><person>Pooh Bear</person></to>
            <order><quantity>1</quantity>
              <item>Pooh Stick</item></order>
          </n:purchaseOrder>
       </s:Body>
    </s:Envelope>
```

RPC/encoded-style SOAP Message

```
public Float getQuote(String symbol);

<s:Envelope
  xmlns:s=http://www.w3.org/2001/06/soap-envelope>
   <s:Header>
      <m:transaction xmlns:m=“soap-transaction”
               s:mustUnderstand=“true”>
         <transactionID>1234</transactionID>
      </m:transaction>
   </s:Header>
   <s:Body>
      <n:getQuote xmlns:n=“http://example/QuoteService.wsdl”>
         <symbol xsi:type=“xsd:string”>IBM</symbol>
      </n:getQuote>
   </s:Body>
</s:Envelope>
```

SOAP RPC Response

```
<s:Envelope
     xmlns:s=http://www.w3.org/2001/06/soap-envelope>
    <s:Body>
      <n:getQuoteResponse
      xmnls:n="http://example/QuoteService.wsdl">
        <value xsi:type="xsd:float">
          98.06
        </value>
      </n:getQuoteResponse>
    </s:Body>
  </s:Envelope>
```

SOAP Faults

- SOAP has a model for errors
- Distinguishes between
  - Detecting faults
  - Signalling faults
- Upon the detection of a fault, a fault message is generated
  - env:Fault element is carried in the env:Body
  - Two mandatory sub-elements
    - env:Code
    - env:Reason (human readable)
  - Optional
    - env:Detail, env:Node, env:Role

SOAP 1.2 Faults

```
<s:Envelope xmlns:s="…">
      <s:Body>
        <s:Fault>
        <s:Code>Client.Authentication</s:Code>
        <s:Reason>Invalid credentials</s:Reason>
        <s:Detail>
        <!-- application specific details -->
        </s:Detail>
        </s:Fault>
      </s:Body>
    </s:Envelope>
```

SOAP 1.1 Faults

```
<s:Envelope xmlns:s="…">
    <s:Body>
      <s:Fault>
      <faultcode>Client.Authentication</faultcode>
      <faultstring>Invalid credentials</faultstring>
      <details>
      <!-- application specific details -->
      </details>
      </s:Fault>
    </s:Body>
  </s:Envelope>
```

Standard SOAP Fault Codes

- Version Mismatch
- MustUnderstand: an immediate child element of the header was not understood. Specifies which header blocks were not understood
- Server: server-side processing error
- Client: there is a problem in the message (e.g. incorrectly formed message, invalid authentication credentials, ..)

SOAP 1.2 NotUnderstood Header

```
<env:Envelope xmlns:env="…">
    <env:Header>
        <env:NotUnderstood qname="t:transaction" xmlns:t=http://../>
    </env:Header>
    <env:Body>
        <env:Fault>
        <env:Code><env:Value>env:MustUnderstand</env:Value>
        </env:Code>
        <env:Reason>
        <env:Text xml:lang="en-US">Header not
        understood</env:Text>
        </env:Reason>
        </env:Fault>
    </env:Body>
</env:Envelope>
```

SOAP 1.1 Misunderstood Header

```
<s:Envelope
 xmlns:s=http://www.w3.org/2001/06/soap-envelope>
  <s:Header>
    <f:Misunderstood qname="abc:transaction"
       xmlns:="soap-transactions" />
  </s:Header>
  <s:Body>
    <s:Fault>
      <faultcode>MustUnderstand</faultcode>
      <faultstring>
     Header(s) not understood
   </faultstring>
      <faultactor>http://acme.com/</faultactor>
    </s:Fault>
  </s:Body>
</s:Envelope>
```

XML Routing

- SOAP may be used to route XML documents in a distributed system
- In content-based routing the contents of the SOAP document (or an XML-document) are used in making the forwarding decision
  - XPath statements on header / body
- SOAP does not define a message path in itself
  - WS-Addressing (Recommendation Core + SOAP Binding)
- Performance issues for SOAP processing:
  - transport protocol
  - on-the-wire representation (some commercial systems use gzip)
  - in-memory-representation, SAX has less overhead than DOM (DOM is not suitable for streaming)
  - in-memory processing, how is the XML tree accessed and matched

WS-Addressing

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
  <S:Header>
    <wsa:MessageID>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://business456.example/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
    <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
  </S:Header>
  <S:Body>
    ...
  </S:Body>
</S:Envelope>
```

Wireless SOAP I

- SOAP is seen as promising for wireless environments because of its interoperability
- However, SOAP implementations have several limitations:
  - HTTP is not efficient in wireless environments
    - TCP is not efficient
    - HTTP has overhead
  - XML data has overhead both on-the-wire and in-memory
    - XML 1.0 representation is not suitable for low-bandwidth, high-latency links
  - Request-reply semantics may slow down applications

Wireless SOAP II

- Current research at HIIT addresses these issues:
  - Efficient message transport protocol for SOAP
  - Event stream based representation for efficient on-the-wire transmission and in-memory operations
  - Event stream compression and token omission
  - Schema-based compression
- Currently there is a W3C Working Group for addressing bit-efficient XML representation
  - Efficient XML Interchange WG
- SOAP for small devices:
  - kXML and kSOAP
  - J2ME Web Services 1.0 JSR-172
    - JAX-RPC Java API for interacting with SOAP
    - JAX-M JSR-67

SOAP Summary

- SOAP is a one-way and request-reply communication protocol for exchanging messages between decentralized peers
- SOAP is based on the XML Infoset
  - Allows different on-the-wire representations
  - Support for custom data types and custom encoding rules
- SOAP 1.2 is a W3C Recommendation
- SOAP header mechanism allows routing of XML documents and supports intermediaries
- SOAP is becoming increasingly popular
  - .NET, Sun J2EE, Apache Axis, Google,..
- And it is being extended for the wireless environment
- Current challenge
  - Portability and interoperability across implementations

UDDI

- Universal Description Discovery and Integration
- A “meta service” for locating web services by enabling robust queries against rich metadata
- Distributed registry of businesses and their service descriptions implemented in a common XML format

Web Service Challenges

- Who provides web services?
- How are they implemented?
- Where are they provided?
- What is their behavior?
- Is an application compatible?
- Searching and indexing do not work today
  - how to find the right services at the right time?
  - we need solutions designed for tools and apps

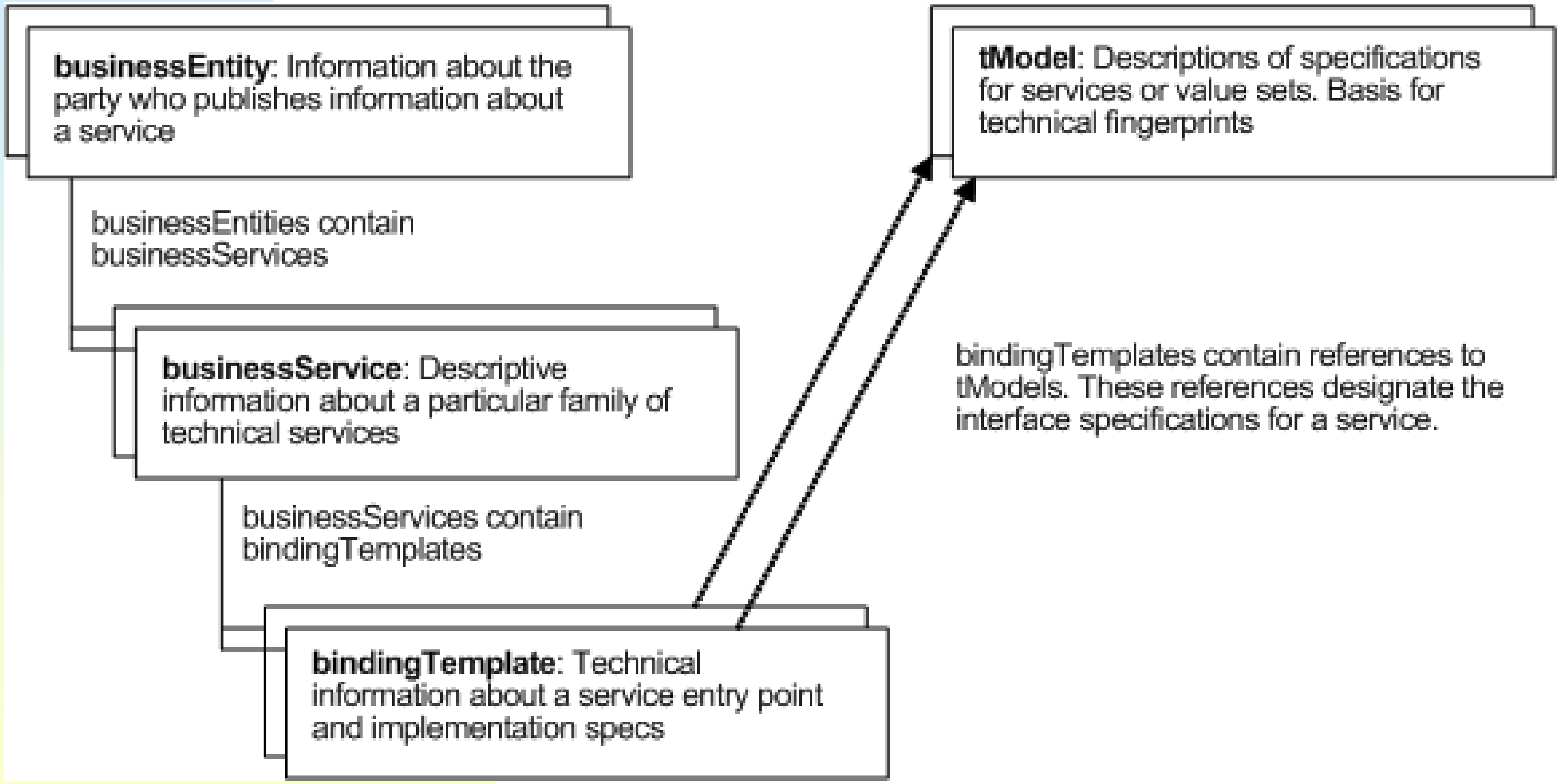
What is UDDI?

- Universal Description Discovery and Integration
- Industry-wide initiative supporting web services
- Specifications
  - Schemas for service description
  - Schemas for business (service implementers) description
  - Developed on industry standards
    - Applies equally to XML and non-XML web services
- Implementation
  - Public web service registry and development resources
  - SOAP-based programming protocol for registering and discovering Web services
    - XML schema for SOAP messages
    - a description of the API
- UDDI does not directly specify how pricing, deadlines, etc. are handled/matched
  - Advanced discovery via portals and marketplaces

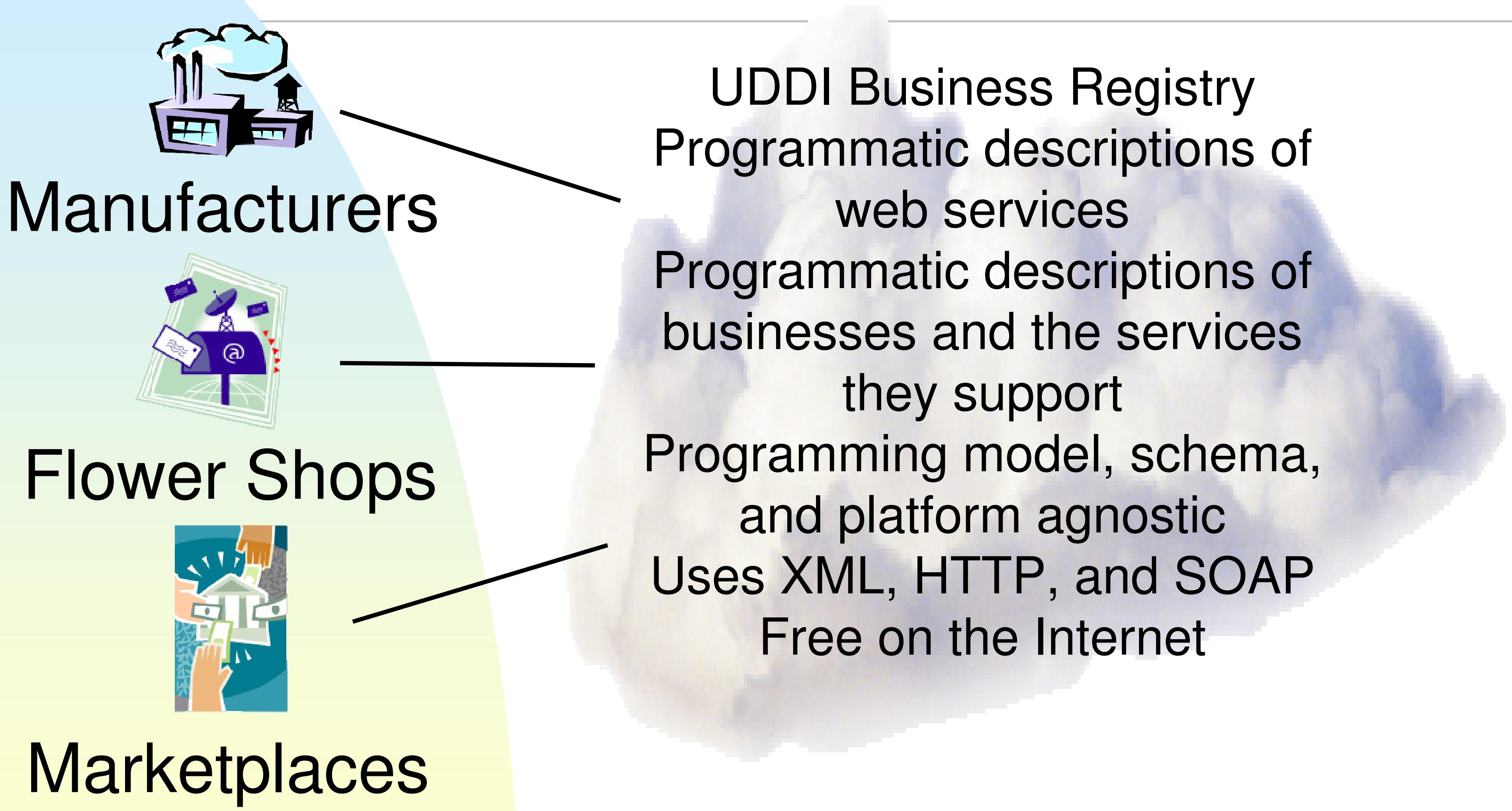
Again: What is UDDI?

- A project to speed interoperability and adoption for web services
  - Standards-based specifications for service description and discovery
  - Shared operation of a business registry on the web
- Partnership among industry and business leaders
- With UDDI a programmer or a program can locate
  - Information about services exposed by a partner
  - find compatible in-house services
  - Find links to specifications of a Web service
  - Maintain technical compatibility by automatically configuring certain technical connections
- Businesses can locate potential partners

The four core types of data structures that are specified by the UDDI API Schema and their relationships are shown here


businessEntity: Information about the party who publishes information about a service
businessEntities contain businessServices
businessService: Descriptive information about a particular family of technical services
businessServices contain bindingTemplates
bindingTemplate: Technical information about a service entry point and implementation specs
tModel: Descriptions of specifications for services or value sets. Basis for technical fingerprints
bindingTemplates contain references to tModels. These references designate the interface specifications for a service.

UDDI v1 Implementation



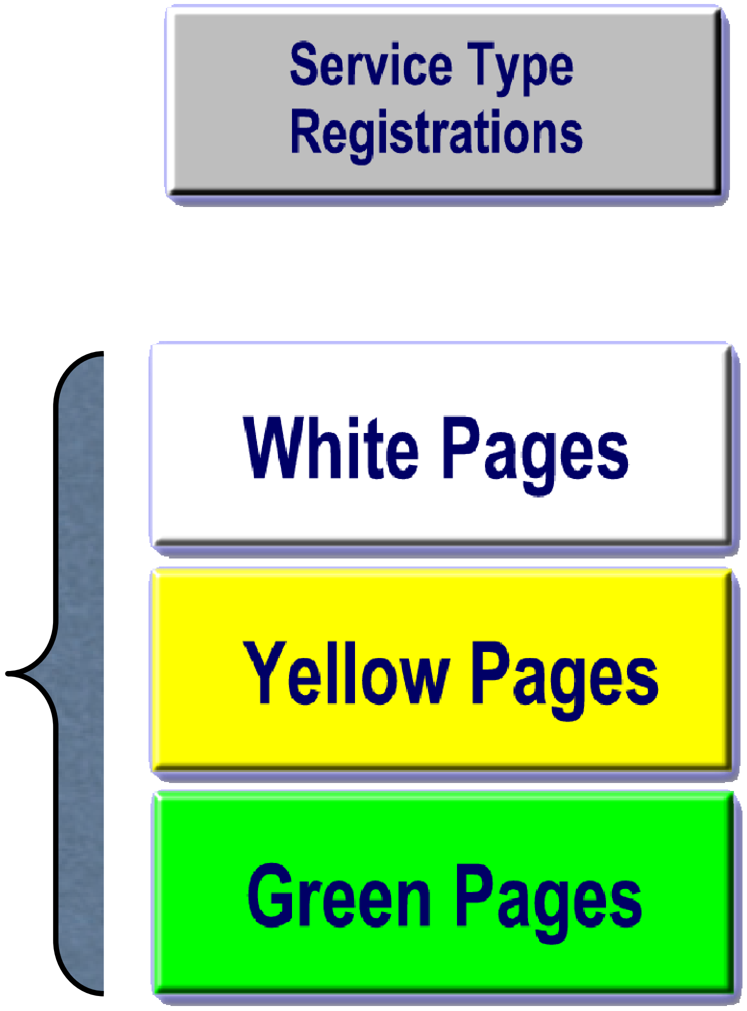
UDDI Business Registry
Programmatic descriptions of web services
Programmatic descriptions of businesses and the services they support
Programming model, schema, and platform agnostic
Uses XML, HTTP, and SOAP
Free on the Internet

Source: www.uddi.org, UDDI Overview presentation 9/6/2000

UDDI Registry Entries

Standards Bodies, Agencies, Programmers, Publishers register specifications for their Service Types

Service providers register precise information about themselves and their Web services



White pages

- Business name
- General business description
  - Any number of languages
- Contact info
  - Names, phone numbers, fax numbers, web sites, etc.
- Known identifiers
  - List of unique identifiers for a business

Yellow pages

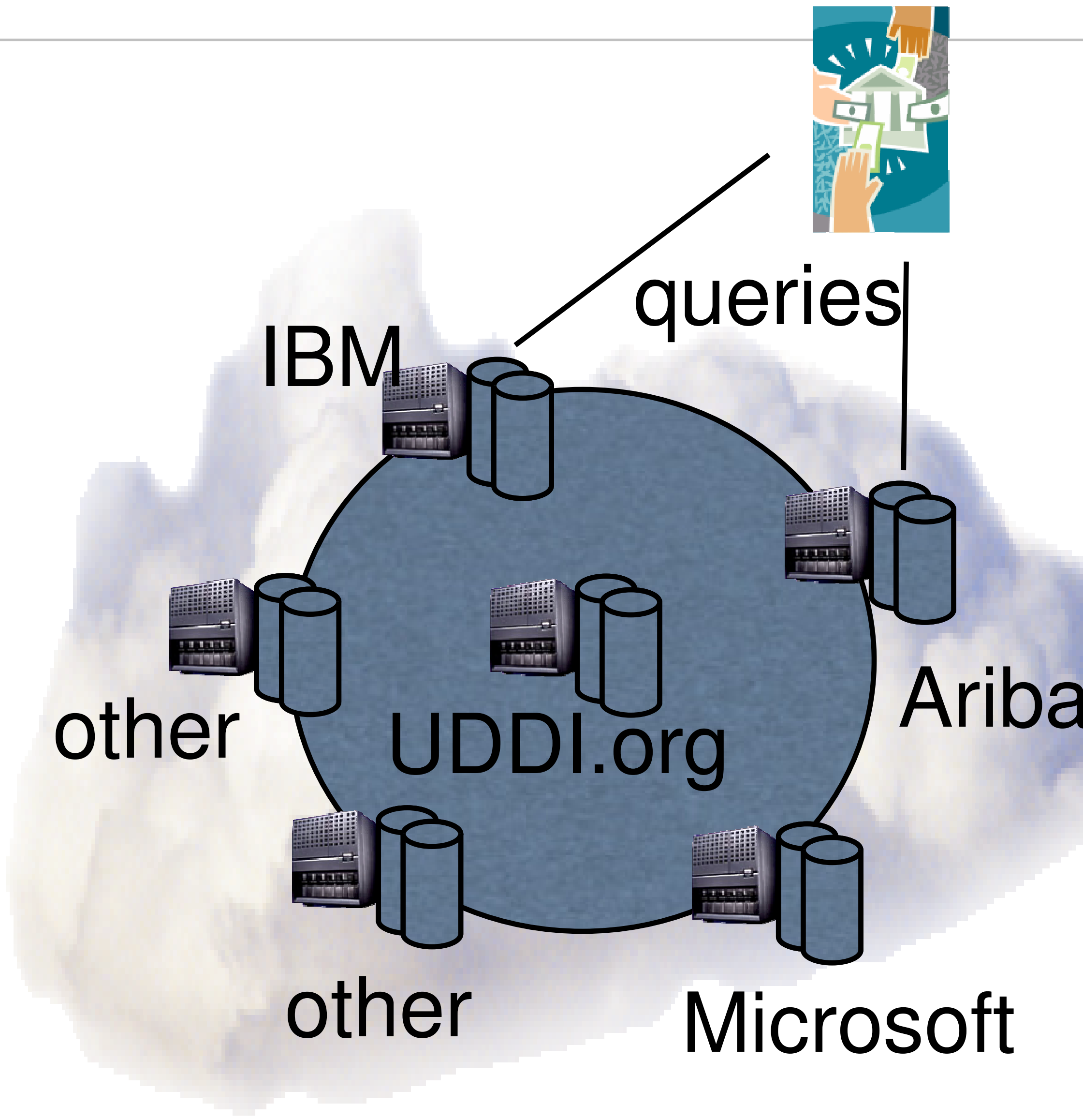
- Business categories
  - Based on standard taxonomies
  - 3 base taxonomies in V1
- Taxonomies
  - Industry: NAICS (Industry codes - US Govt.)
  - Product/Services: UNSPSC (ECCMA)
  - Location: Geographical taxonomy (ISO 3166)
  - …easy extension in upcoming releases
- Realized using name-value pairs, any valid taxonomy identifier can be attached to the business white page

Green pages

- New set of information businesses use to describe how to “do e-commerce” with them
  - References to specifications for Web Services
- Business process (functional)
- Service specifications (technical)
  - Programming/platform/implementation agnostic
- Binding information (implementation)

Now look at that again:Registry Operation

- Peer nodes (websites)
- 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
- Compliance enforced by business contract



Source: www.uddi.org, UDDI Overview presentation 9/6/2000

The programmer's API Implementation

- UDDI is up and running at Microsoft, IBM, and Ariba.
- An online Web Service that you can use from your applications to dynamically discover other online services, all neatly packaged in a simple XML interface:
  - http://uddi.microsoft.com/inquire
  - http://uddi.ariba.com/UDDIProcessor.aw/ad/process
  - http://www-3.ibm.com/services/uddi/inquiryapi
- These are the UDDI entry points for "INQUIRIES". The entry points for updates are different and are typically HTTPS addresses for security reasons.

UDDI Invocation Model

1. The programmer uses the UDDI business registry to locate the businessEntity information for the desired advertised Web Service
2. The programmer selects a particular bindingTemplate and saves it
3. The program is prepared on this knowledge, obtained from tModel key information in the bindingTemplate
4. At runtime, the program invokes the Web service as planned using the cached bindingTemplate information

Registry APIs

- Inquiry API
  - find_business, find_service, find_binding, find_tModel
  - get_businessDetail, get_serviceDetail, get_bindingDetail, get_tModelDetail
- Publisher's API
  - save_business, save_service, save_binding, save_tModel
  - delete_business, delete_service, delete_binding, delete_tModel
- Security
  - get_authToken, discard_authToken

What XML Do You POST?

```
<?xml version='1.0' encoding='UTF-8'?>
      <Envelope
     xmlns='http://schemas.xmlsoap.org/soap/envelope/'>
        <Body>
    <find_business generic="1.0"
          xmlns="urn:uddi-org:api">
           <name>Microsoft</name>
          </find_business>
   </Body>
      </Envelope>
```

How Do You Post the XML?

```
http = new
ActiveXObject("Microsoft.XMLHTTP");
http.open("POST", url, false);
http.setRequestHeader("Accept","text/xml");
http.setRequestHeader("Cache-Control","no-
cache");
http.setRequestHeader("SOAPAction","\"\"");
http.send(msg);
```

What Do You Get Back?

```
<businessList generic="1.0"
   operator="Microsoft Corporation"
      truncated="false" xmlns="urn:uddi-org:api">
    <businessInfos>
      <businessInfo
   businessKey="0076B468-EB27-42E5-AC09-9955CFF462A3">
      <name>Microsoft Corporation</name>
   <serviceInfos>
        <serviceInfo
      businessKey="0076B468-…-9955CFF462A3"
         serviceKey="8BF2F51F-…-38D8205D1333">
          <name>EBI Services</name>
        </serviceInfo>
        <serviceInfo
      businessKey="0076B468-…-9955CFF462A3"
         serviceKey="D2BC296A-…-494F9E53F1D1">
          <name>UDDI Web Services</name>
        </serviceInfo>
```

More Information

- UDDI Resources
  - http://www.uddi.org
  - http://uddi.microsoft.com
  - http://www-3.ibm.com/services/uddi
- For Developers
  - SOAP/Web Services SDK
  - Visual Basic UDDI SDK
  - IBM AlphaWorks Web Services Toolkit