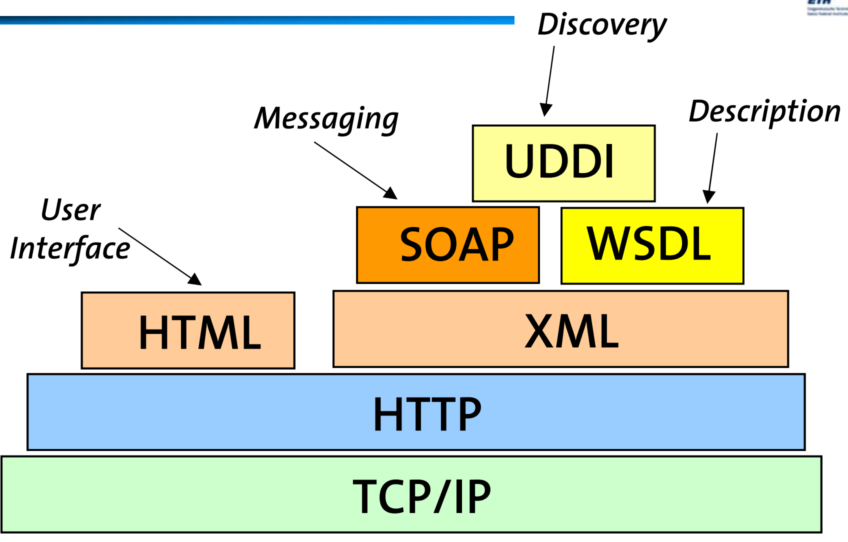
Distributed Systems
UDDI and beyond

Dr. Cesare Pautasso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
pautasso@inf.ethz.ch
http://www.inf.ethz.ch/~pautasso

Where we are



Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Universal Description, Discovery and Integration (UDDI)

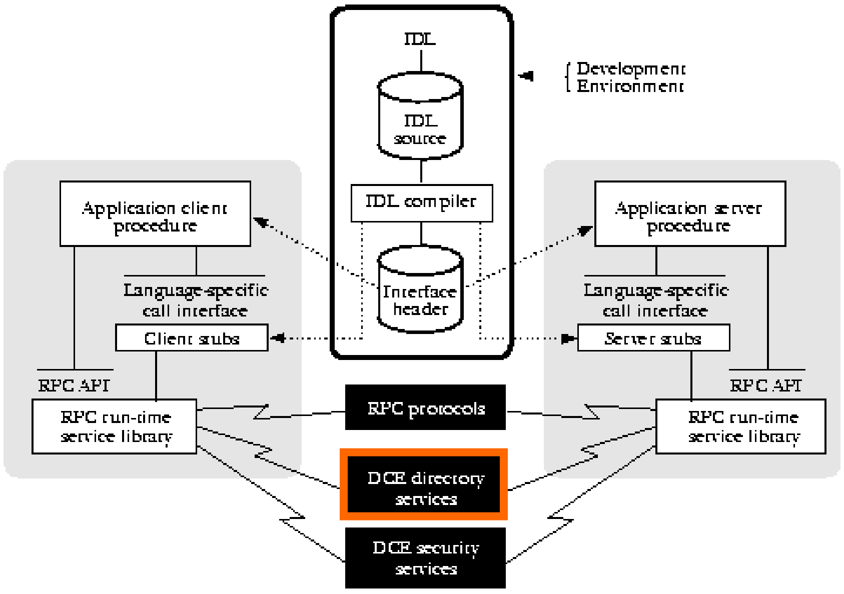
Basic Problems to solve

1. How to make the service invocation part of the language in a more or less transparent manner.
   - Don't forget this important aspect: whatever you design, others will have to program and use
2. How to exchange data between machines that might use different representations for different data types. This involves two aspects:
   - data type formats (e.g., byte orders in different architectures)
   - data structures (need to be flattened and the reconstructed)

UDDI

3. How to find the service one actually wants among a potentially large collection of services and servers.
   - The goal is that the client does not necessarily need to know where the server resides or even which server provides the service.
4. How to deal with errors in the service invocation in a more or less elegant manner:
   - server is down,
   - communication is down,
   - server busy,
   - duplicated requests ...

DCE architecture

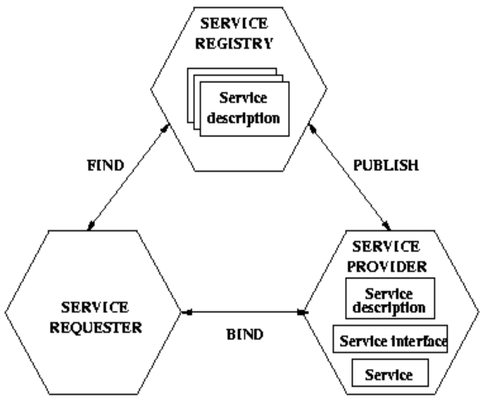

IDL
IDL source
IDL compiler
Interface header
Development Environment
Application client procedure
Language-specific call interface
Client stubs
RPC API
RPC run-time service library
Application server procedure
Language-specific call interface
Server stubs
RPC API
RPC run-time service library
RPC protocols
DCE directory services
DCE security services

What is UDDI?

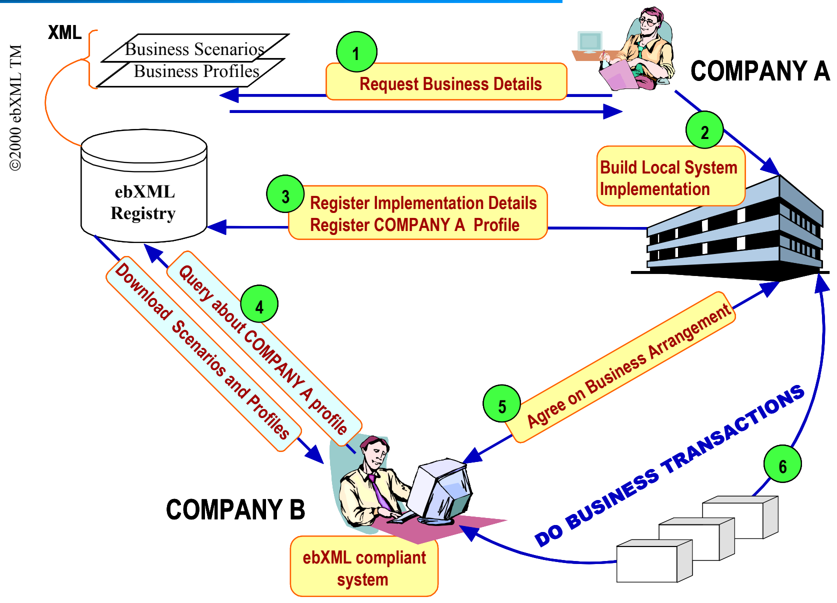
- The UDDI specification is probably the one that has evolved the most from all specifications we have seen so far. The latest version is version 3 (July 2002):
  - version 1 defined the basis for a business service registry
  - version 2 adapted the working of the registry to SOAP and WSDL
  - version 3 redefines the role and purpose of UDDI registries, emphasizes the role of private implementations, and deals with the problem of interaction across private and public UDDI registries
- Originally, UDDI was conceived as an “Universal Business Registry” similar to search engines (e.g., Google) which will be used as the main mechanism to find electronic services provided by companies worldwide. This triggered a significant amount of activity around very advanced and complex scenarios (Semantic Web, dynamic binding to partners, runtime/automatic partner selection, etc.)
- Nowadays UDDI is far more pragmatic and recognizes the realities of B2B interactions: it presents itself as the “infrastructure for Web services”, meaning the same role as a name and directory service (i.e., binder in RPC) but applied to Web services and mostly used in constrained environments (internally within a company or among a predefined set of business partners)

Role of UDDI

- Services offered through the Internet to other companies require much more information that a typical middleware service
- In many middleware and EAI efforts, the same people develop the service and the application using the service
- This is obviously no longer the case and, therefore, using a service requires much more information that it is typically available for internal company services
- This documentation has three aspects to it:
  - basic information
  - categorization
  - technical data



More detailed (ebXML architecture)


©2000 ebXML TM
XML
Business Scenarios
Business Profiles
1
Request Business Details
COMPANY A
2
Build Local System Implementation
ebXML Registry
3
Register Implementation Details
Register COMPANY A Profile
Download Scenarios and Profiles
Query about COMPANY A profile
4
5
Agree on Business Arrangement
DO BUSINESS TRANSACTIONS
6
COMPANY B
ebXML compliant system

Information in an UDDI registry

UDDI data

- An entry in an UDDI registry is an XML document composed of different elements (labeled as such in XML), the most important ones being:
  - *businessEntity* : is a description of the organization that provides the service.
  - *businessService*: a list of all the Web services offered by the business entity.
  - bindingTemplate: the technical aspects of the service being offered.
  - *tModel*: (“technical model”)is a generic element that can be used to store addotional information about the service, typically additional technical information on how to use the service, conditions for use, guarantees, etc.
- Together, these elements are used to provide:
  - white pages information: data about the service provider (name, address, contact person, etc.)
  - yellow pages information: what type of services are offered and a list of the different services offered
  - green pages information: technical information on how to use each one of the services offered, including pointers to WSDL descriptions of the services (which do not reside in the UDDI registry)

Business entity

- The generic white and yellow pages information about a service provider is stored in the businessEntity, which contains the following data:
  - each businessEntity has a businessKey
  - discoveryURLs: a list of URLs that point to alternate, file based service discovery mechanisms.
  - Name: (textual information)
  - Business description: (textual information)
  - Contacts: (textual information)
  - businessServices: a list of services provided by the businessEntity
  - identifierBag: a list of external identifiers
  - categoryBag: a list of business categories (e.g., industry, product category, geographic region)
- The businessEntity does not need to be the company. It is meant to represent any entity that provides services: it can be a department, a group of people, a server, a set of servers, etc

Business service

- The services provided by a business entity re described in business terms using businessService elements. A businessService element can describe a single Web service or a group of related Web services (all of them offered by the same businessEntity)
- A businessEntity can have several businessServices but a businessService belongs to one businessEntity
- The businessService can actually by provided by a different businessEntity that the one where the element is found. This is called projection and allows to include services provided by other organizations as part of the own services
- It contains:
  - a serviceKey that uniquely identifies the service and the businessEntity (not necessarily the same as where the businessService is found)
  - name: as before
  - description: as before
  - categoryBag: as before
  - bindingTemplates: a list to all the bindingTemplates for the service with the technical information on how to access and use the service

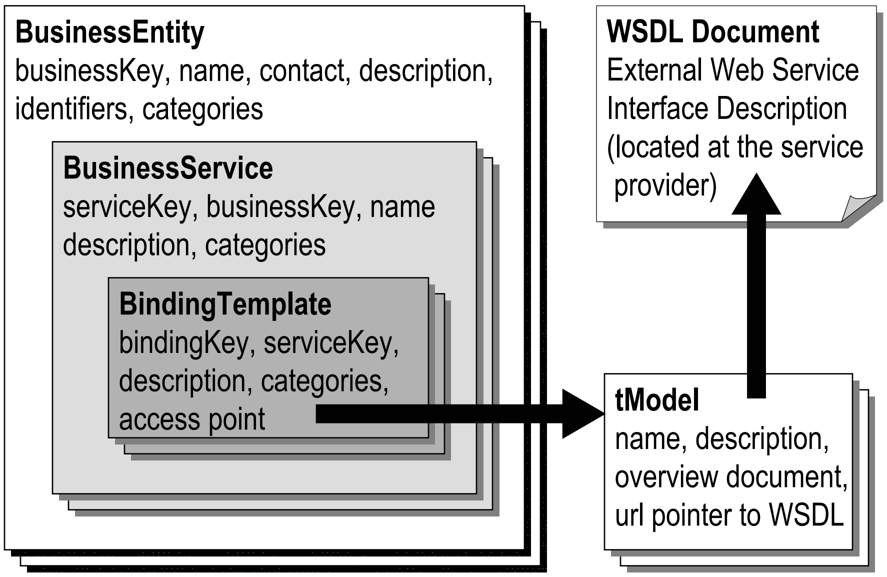
Binding template

- A binding template contains the technical information associated to a particular service. It contains the following information:
  - bindingKey
  - serviceKey
  - description
  - accessPoint: the network address of the service being provided (typically an URL but it can be anything as this field is a string: e.g., an e-mail address or even a phone)
  - tModels: a list of entries corresponding to tModels associated with this particular binding. The list includes references to the tModels, documents describing these tModles, short descriptions, etc.
  - categoryBag: additional information about the service and its binding (e.g., whether it is a test binding, it is on production, etc)
- A businessService can have several bindingTemplates but a binding Tenplate has only one businessService
- The binding template can be best seen as a folder where all the technical information of a service is put together

tModel

- A tModel is a generic container of information where designers can write any techical information associated to the use of a Web service:
  - the actual interface and protocol used, including a pointer to the WSDL description
  - description of the business protocol and conversations supported by the service
- A tModel is a document with a short description of the technical information and a pointer to the actual information. It contains:
  - tModelKey
  - name
  - description
  - overviewDoc: (with an overviewURL and useType that indicate where to find the information and its format, e.g., “text” or “wsdldescription”)
  - identifierBag
  - categoryBag
- A tModel can point to other tModels and eventually different forms of tModels will be standardized (tModel for WSDL services, tModels for EDI based services, etc.)

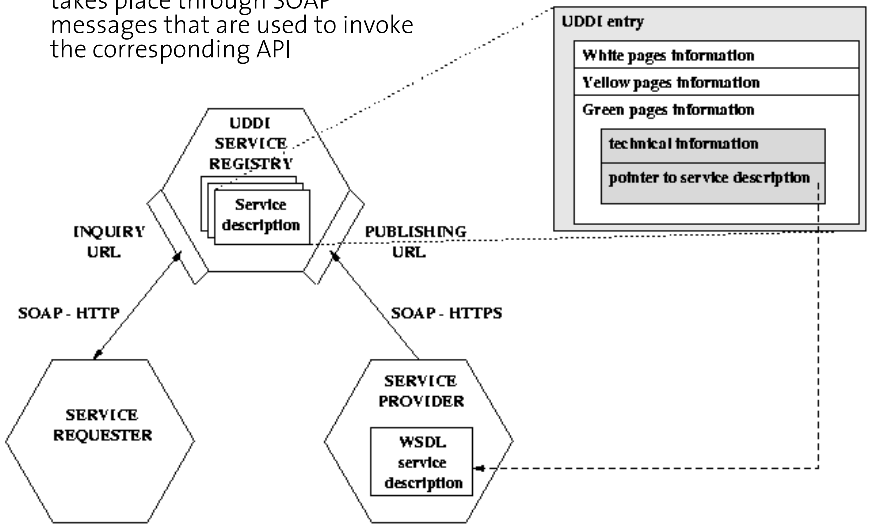
Summary of the UDDI data model


BusinessEntity
businessKey, name, contact, description, identifiers, categories
BusinessService
serviceKey, businessKey, name description, categories
BindingTemplate
bindingKey, serviceKey, description, categories, access point
WSDL Document
External Web Service Interface Description (located at the service provider)
tModel
name, description, overview document, url pointer to WSDL

Interacting with an UDDI registry

Inquiry and Publishing interfaces

- Access to an UDDI registry typically takes place through SOAP messages that are used to invoke the corresponding API



UDDI interfaces

- The UDDI specification provides a number of Application Program Interfaces (APIs) that provide access to an UDDI system:
  - UDDI Inquiry: to locate and find details about entries in an UDDI registry. Support a number of patterns (browsing, drill-down, invocation)
  - UDDI Publication: to publish and modify information in an UDDI registry. All operations in this API are atomic in the transactional sense
  - UDDI Security: for access control to the UDDI registry (token based)
  - UDDI Subscription: allows clients to subscribe to changes to information in the UDDI registry (the changes can be scoped in the subscription request)
  - UDDI Replication: how to perform replication of information across nodes in an UDDI registry
  - UDDI Custody and Ownership transfer: to change the owner (publisher) of information and ship custody from one node to another within an UDI registry
- UDDI also provides a set of APIs for clients of an UDDI system:
  - UDDI Subscription Listener: the client side of the subscription API
  - UDDI Value Set: used to validate the information provided to an UDDI registry

UDDI inquiry API

- Search and lookup entries in a registry.
- This API is freely available, no client authentication is required.
- Errors are reported as SOAP Faults
- Browse functions search the registry based on keywords and return summary lists with overview information (key, name and description) about matching businesses or services.
- Find qualifiers are used to sort the results and to control the keyword matching: toggle between AND/OR, case sensitive/insensitive, use of wildcards and categories.
- To minimize the number of requests, find queries can be nested
- Drill-down functions are used to fetch the specific UDDI data structures about particular entries given their key, returned by the Browse functions

Browse functions
find_business
find_relatedBusinesses
find_service
find_binding
find_tModel

Drill down functions
get_businessDetail
get_operationalInfo
get_serviceDetail
get_bindingDetail
get_tModelDetail

UDDI Version 3.0 Specification, 19 July 2002

UDDI publishing and security API

- Publish, update and delete information contained in a UDDI registry
- The publishing API requires user authentication using a session token and typically uses SOAP over HTTPS
- The registry performs access control for all publishing functions: information about the entries can only be edited by the owner
- Category information and keyed references associated to the entries are validated before accepting new information into the registry
- Deletion functions are used to remove entries identified by their key from the registry. Removing a business will remove all services associated with it.
- The same publishing functions are used both to add new information or replace existing information, depending on whether a valid key is passed or not.
- When adding new entries, keys are usually automatically generated by the registry

Security Session Management
get_authToken, discard_authToken

Publishing	Deletion
save_business	*delete_business*
save_service	*delete_service*
save_binding	*delete_binding*
save_tModel	*delete_tModel*

UDDI Summary

- The UDDI specification is rather complete and encompasses many aspects of an UDDI registry from its use to its distribution across several nodes and the consistency of the data in a distributed registry
- Most UDDI registries are private and typically serve as the source of documentation for integration efforts based on Web services
- UDDI registries are not necessarily intended as the final repository of the information pertaining Web services. Even in the "universal" version of the repository, the idea is to standardize basic functions and then built proprietary tools that exploit the basic repository. That way it is possible to both tailor the design and maintain the necessary compatibility across repositories
- While being the most visible part of the efforts around Web services, UDDI is perhaps the least critical due to the complexities of B2B interactions (establishing trust, contracts, legal constrains and procedures, etc.) . The ultimate goal is, of course, full automation, but until that happens a long list of problems need to be resolved and much more standardization is necessary.

References

- Specifications:
  http://www.uddi.org
  http://uddi.org/pubs/uddi-v3.00-published-20020719.pdf
- UDDI Business Registry (UBR) nodes:

IBM
Homepage: http://uddi.ibm.com/
Inquiry API:
http://uddi.ibm.com/ubr/inquiryapi
Publish API:
https://uddi.ibm.com/ubr/publishapi

Microsoft
Homepage: http://uddi.microsoft.com/
Inquiry API:
http://uddi.microsoft.com/inquire
Publish API :
https://uddi.microsoft.com/publish

SAP
Homepage: http://uddi.sap.com/
Inquiry API :
http://uddi.sap.com/uddi/api/inquiry
Publish API :
https://uddi.sap.com/uddi/api/publish

NTT
Homepage:
http://www.ntt.com/uddi/
Inquiry API :
http://www.uddi.ne.jp/ubr/inquiryapi
Publish API :
https://www.uddi.ne.jp/ubr/publishapi

Hype and reality

- There are a few universal UDDI registries in operation (maintained by IBM, Microsoft, SAP, etc)
- These registries are very visible and often the first thing one sees of Web services
- Unfortunately, these registries are still very small and most of the entries in them do not work or do not correspond to any real service
- This has been a source of criticism to We services in general. The criticism has not been entirely undeserved but it is often misguided: what was there to criticize was not UDDI itself but the use that was been made of it and the hype around dynamic Web services
- UDDI is rather useful if seen as supporting infrastructure for Web services in well defined and constrained environments (i.e., without public access and where there is a context that provides the missing information)
- Most of the UDDI registries in place today are private registries operating inside companies (recall that the widest use of Web services today is for conventional EAI) or maintained by a set of companies in a private manner
- UDDI has now become the accepted way to document Web services and supply the information missing in WSDL descriptions

Limitations of UDDI

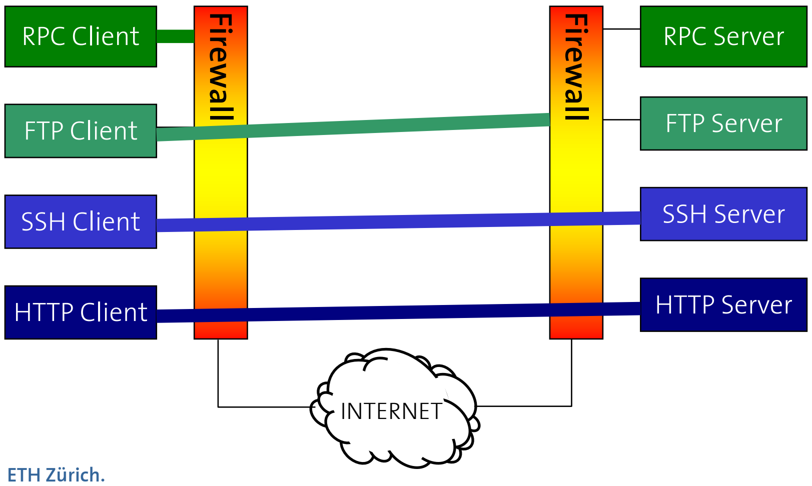
- UDDI was initially proposed as a standard to enable universal discovery of services
- Public registries compliant to UDDI didn't grow as expected:
  - Registries assumed voluntary registration of service providers (registry is passive, as opposed to actively crawling the Web looking for WSDL definitions of services)
  - Registries didn't provide any value-added service, such as checking the quality of the registered services
- Web-based registries such as the XMethods.com portal offer a human-oriented registry of Web services (also based on voluntary registration)
- WSDL Search Engines (like Woogle) offer a simpler query interface that makes it easier to search for services that match a specific interface template (UDDI is quite complex in that regard)
- In addition to service lookup functionality, automatic B2B Integration requires additional capabilities (which are not part of UDDI) such as:
  - Provider Validation
  - Semantics Lookup
  - Quality of Service Metadata
  - Service Level Agreements
  - Contract Negotiation
  - Trust Establishment

Limitations of SOAP, WSDL and UDDI

SOAP and Security

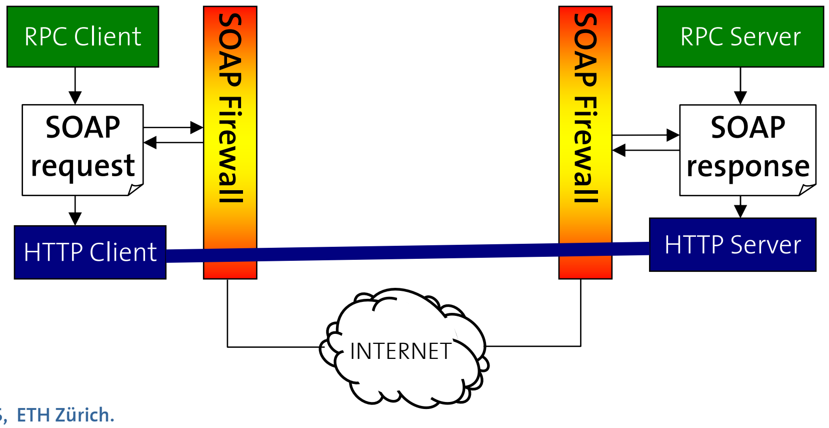
SOAP and Firewalls

- One of the perceived advantages of SOAP lies in its ability to tunnel RPC calls through firewalls
- This works only because SOAP uses HTTP and firewalls do not typically block TCP port 80, the default one used by the HTTP protocol.



SOAP Firewalls?

- Once all communication traffic is encoded in SOAP and sent over HTTP, traditional TCP-level firewalls do not offer an acceptable level of protection because RPC services which were hidden behind specific TPC port numbers are now exposed as SOAP Web services.
- Thus, firewalls become more complex as they must allow or disallow HTTP connections based on the content of the SOAP messages



WS-Security

- The SOAP standard does not make any provision for secure message exchange. As its name implies SOAP is meant to be a simple (but extensible) messaging protocol, and properties such as security can be added to it.
- WS-Security is a SOAP extension that addresses some of the security issues such as:
  - Message integrity (guarantee that a message is not tampered with)
  - Message confidentiality (guarantee that the content of a message is kept secret)
  - Sender authentication (identify the sender of the message)
- WS-Security prescribes how to use SOAP header blocks to store the digital signature of the message, as well as user identification information and passwords.
- WS-Security enables end-to-end secure message exchange, whereas SOAP on top of HTTPS only guarantees security across each hop.
- With SOAP/HTTPS messages are decrypted and re-encrypted by each intermediate receiver and there is no way to encrypt the SOAP message all the way between initial sender and ultimate receiver
- WS-Security also supports encryption of only specific blocks of a SOAP message (e.g., the ones carrying sensitive information, such as Credit Card numbers)

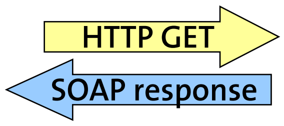
SOAP and Client/Server architectures

SOAP and the client server model

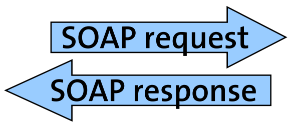
- The close relation between SOAP, RPC and HTTP has two main reasons:
  - SOAP has been initially designed for client server type of interaction which is typically implemented as RPC or variations thereof
  - RPC, SOAP and HTTP follow very similar models of interaction that can be very easily mapped into each other (and this is what SOAP has done)
- The advantages of SOAP arise from its ability to provide a universal vehicle for conveying information across heterogeneous middleware platforms and applications. In this regard, SOAP will play a crucial role in enterprise application integration efforts in the future as it provides the standard that has been missing all these years
- The limitations of SOAP arise from its adherence to the client server model:
  - data exchanges as parameters in method invocations
  - rigid interaction patterns that are highly synchronous
- and from its simplicity:
  - SOAP is not enough in a real application, many aspects are missing

SOAP Message Exchange Patterns

SOAP response



SOAP request-response

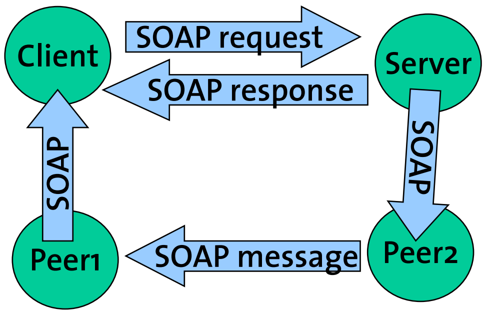


- It involves a request which is not a SOAP message (implemented as an HTTP GET request method which eventually includes the necessary information as part of the requested URL) and a response that is a SOAP message
- This pattern excludes the use of any header information (as the request has no headers)

- It involves sending a request as a SOAP message and getting a second SOAP message with the response to the request
- This is the typical mode of operation for most Web services and the one used for mapping RPC to SOAP.
- This exchange pattern is also the one that implicitly takes advantage of the binding to HTTP and the way HTTP works

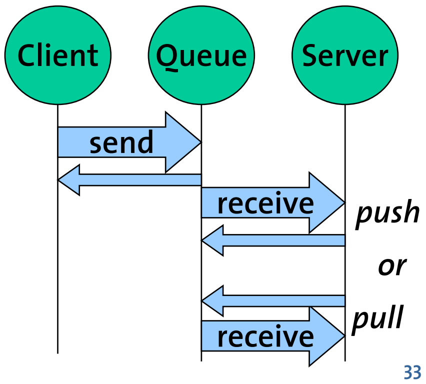
RPC vs. One Way Messaging

- Both of the previous exchange patterns are used to implement a synchronous client/server message exchange, which is just a particular case of more complex message exchange patterns.

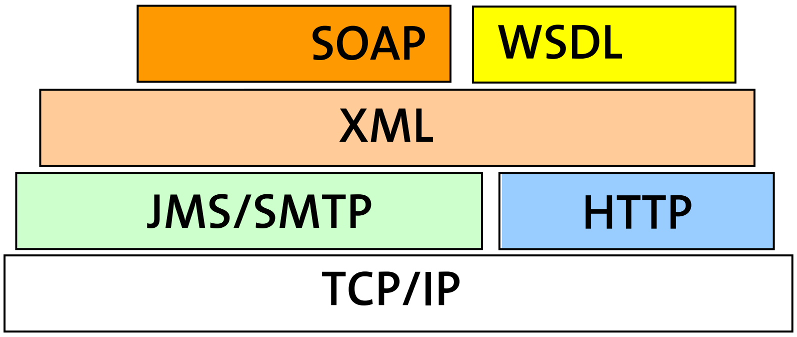


- SOAP messages, however, can also be used as part of asynchronous interactions between a set of peers

- By using techniques developed as part of traditional Message Oriented Middleware, asynchronous messaging can be built on top of synchronous interactions, by introducing a queuing system that stores and forwards the messages.



Standard Layers



Mapping SOAP to e-mail

- Currently, the SOAP specifications (including 1.2) do not contain an e-mail (SMTP) binding, they just show an example of how to send a SOAP message in an e-mail (in 1.2). Two possible options are:
  - as normal e-mail text
  - as an attachment
- In both cases, the SOAP message is not different from what has been discussed so far (in case of HTTP)
- E-mail, however, changes the interaction patterns considered in SOAP (which are very tied to HTTP)
  - SMTP implements a mechanism whereby an e-mail message is automatically responded to with a delivery notification
  - SOAP cannot use the delivery notification message to return the response to the request since the delivery notification message happens at the level of SMTP, not at the level of the SOAP protocol
  - the current 1.2 draft warns about the limitations of e-mail binding for SOAP reflecting once more the implicit client server model that inspires the design and development of SOAP

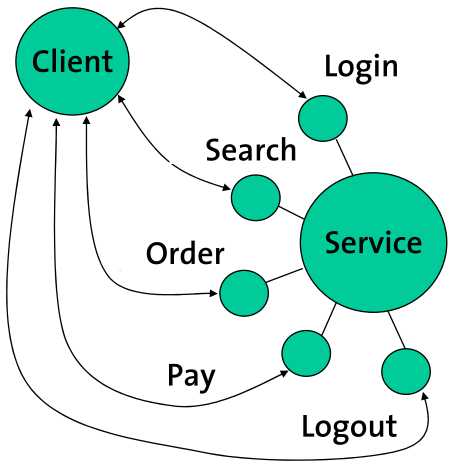
IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Conversations

Conversations

- As a first approximation, a conversation models the sequences of operations that a client may invoke as part of the interaction with a Web service.
- In general, a conversation defines a complex interaction between multiple Web services involving the exchange of several messages and the invocation of different operations in a well defined order.
- In this context, a coordination protocol specifies the set of correct conversations between the various services
- The service interface description (WSDL) only lists the available operations but does not specify what is the correct order of invoking them



WSDL and Conversations

- WSDL defines the interface of a Web service in terms of what are the messages that are exchanged (received and produced by the service)
- A WSDL document also structures the messages into pairs (that correspond to the operations provided by a service)
- However, WSDL does not contain any further information specifying what is the correct order of invocation of the various operations. If an operation should not (yet) be invoked, a fault message is returned.
- From the client's point of view, this makes it difficult to automatically ensure the correctness of the interaction.
- On the service side, an interaction across multiple operations may require to maintain session information. (stateful interaction). This information is also used to enforce the correctness of the interaction. Whatever mechanism is employed, these constraints do not surface in the WSDL interface description.
- The goal is to make the development as automatic as possible!

SOAP is XML

The need for Attachments

- SOAP is based on XML and relies on XML for representing data types
- The original idea in SOAP was to make all data exchanged explicit in the form of an XML document much like what happens with IDLs in conventional middleware platforms
- This approach reflects the implicit assumption that what is being exchanged is similar to input and output parameters of program invocations

From SOAP Version 1.2 Part 0: Primer.
© W3C December 2002

```
<env:Body>
  <p:itinerary
   xmlns:p="http://.../reservation/travel">
   <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:depDate>2001-12-14</p:depDate>
    <p:depTime>late afternoon</p:depTime>
<p:seatPreference>aisle</p:seatPreference>
   </p:departure>
   <p:return>
     <p:departing>Los Angeles</p:departing>
     <p:arriving>New York</p:arriving>
     <p:depDate>2001-12-20</p:depDate>
     <p:depTime>mid-morning</p:depTime>
     <p:seatPreference/>
   </p:return>
  </p:itinerary>
</env:Body>
```

- This approach makes it very difficult to use SOAP for exchanging complex data types that cannot be easily translated to XML (and there is no reason to do so): images, binary files, documents, proprietary representation formats, embedded SOAP messages, etc.

A possible solution

- There is a “SOAP messages with attachments note” proposed in 11.12.02 that addresses this problem
- It uses MIME types (like e-mails) and it is based in including the SOAP message into a MIME element that contains both the SOAP message and the attachment (see next page)
- The solution is simple and it follows the same approach as that taken in e-mail messages: include a reference and have the actual attachment at the end of the message
- The MIME document can be embedded into an HTTP request in the same way as the SOAP message
- Problems with this approach:
  - handling the message implies dragging the attachment along, which can have performance implications for large messages
  - scalability can be seriously affected as the attachment is sent in one go (no streaming)
  - not all SOAP implementations support attachments
  - SOAP engines must be extended to deal with MIME types (not too complex but it adds overhead)
- There are alternative proposals like DIME of Microsoft (Direct Internet Message Encapsulation) and WS-attachments

Attachments in SOAP

From SOAP Messages with Attachments. © W3C Note 11 December 2000

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
              type=text/xml;
              start="<claim061400a.xml@claiming-it.com>"
Content-Description: This is the optional message description.
--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: <claim061400a.xml@claiming-it.com>

<?xml version='1.0' ?>
<SOAP-ENV:Envelope
 xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
 ..
 <theSignedForm href="cid:claim061400a.tiff@claiming-it.com"/>
 ..
 </SOAP-ENV:Body>
 </SOAP-ENV:Envelope>
--MIME_boundary
Content-Type: image/tiff
Content-Transfer-Encoding: binary
Content-ID: <claim061400a.tiff@claiming-it.com>

...binary TIFF image...
--MIME_boundary
```

SOAP Message

Reference

ATTACHMENT

The problems with attachments

- Attachments are relatively easy to include in a message and all proposals (MIME or DIME based) are similar in spirit
- The differences are in the way data is streamed from the sender to the receiver and how these differences affect efficiency
  - MIME is optimized for the sender but the receiver has no idea of how big a message it is receiving as MIME does not include message length for the parts it contains
  - this may create problems with buffers and memory allocation
  - it also forces the receiver to parse the entire message in search for the MIME boundaries between the different parts (DIME explicitly specifies the length of each part which can be use to skip what is not relevant)
- All these problems can be solved with MIME as it provides mechanisms for adding part lengths and it could conceivably be extended to support some basic form of streaming
- Technically, these are not very relevant issues and have more to do with marketing and control of the standards
- The real impact of attachments lies on the specification of the interface of Web services (how to model attachments in WSDL?)

IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Integrating Mismatching Services

Syntax and Semantics

- One of the advantages of using self-describing XML for encoding SOAP messages is that it becomes really easy to develop the corresponding parsers (for reading messages) and emitters (for writing messages).
- There are however some disadvantages, not only related to the performance overhead (XML parsing and validation is expensive) but also to the limitations of XML as a data exchange format (SOAP Attachments for exchanging binary data)
- Another problem is that **parseability** does not guarantee **interoperability**.
- The fact that all parties involved can parse SOAP messages, only solves the interoperability problem at the syntax level. Although progress has already been made by standardizing the syntax, there is still a lot to be done to agree on the semantics of the messages.
- At the SOAP-level, it may be necessary to apply transformations to the messages that are exchanged (Data mapping tools for EAI have not disappeared, they have just become XML/XSLT based)
- At the WSDL-level, it should be possible to describe the semantics in addition to the syntax of the service interfaces.

Modeling interface syntax with WSDL

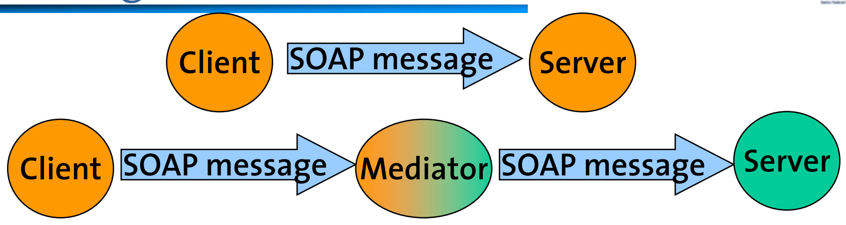
- WSDL defines a service interface (or port type) as a set of operations, grouping together pairs of messages, which are defined in terms of parts (with name and data type, defined in an XML schema).
- From a WSDL description it is possible to automatically infer (and validate) the structure of the corresponding SOAP messages.

```
<message name="getRateRequest">
<part name="country1" type="xsd:string" />
<part name="country2" type="xsd:string" />
</message>
```

<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<n:getRate> <**country1** xsi:type="xsd:string">**USD**</country1>
<**country2** xsi:type="xsd:string">**CHF**</country2> </n:getRate> </soap:Body>

<soap:Body soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<n:getRate> <**country1** xsi:type="xsd:string">**usa**</country1>
<**country2** xsi:type="xsd:string">**switzerland**</country2> </n:getRate>
</soap:Body>

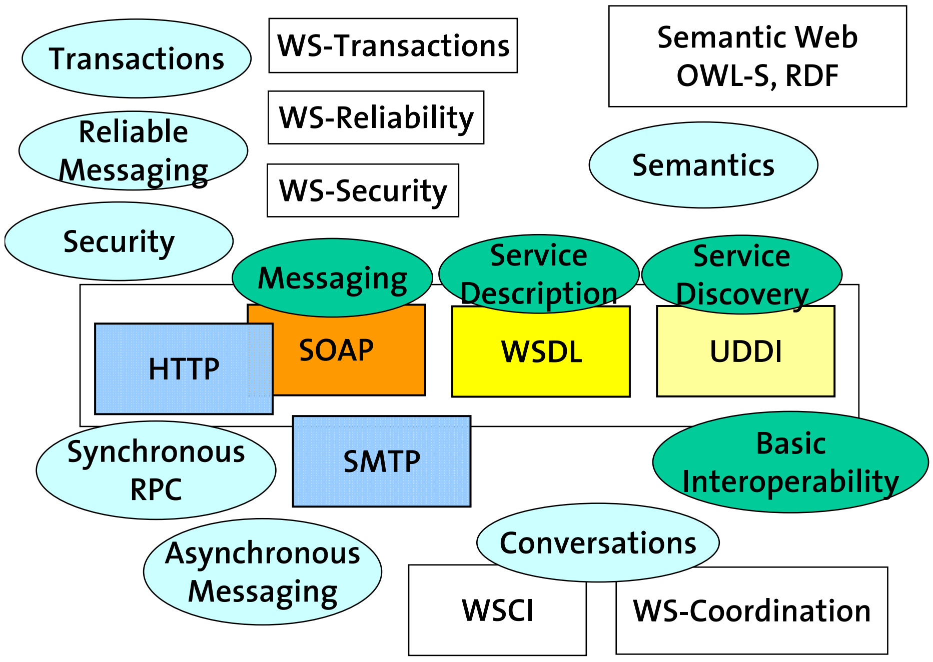
Handling data transformations



- In this example, both client and server use WSDL to describe their interface and SOAP to exchange a message. Even if we assume that the two parties are somehow compatible, this standardization doesn't guarantee interoperability, unless both services use the same XML Schema and (abstracted from the interface description), they agree on the semantics of the message.
- If it is possible to address this mismatch, the message cannot be sent directly, but should be transformed between the two schemas while preserving its semantics.
- This transformation can occur at the client-side (the client knows how to adapt to a given server), at the server-side (the server supports different data models) or – in a true integration scenario – in the middle (using a mediator service)

Modeling interface semantics

- Each syntactical element of a service interface (message, data structure or operation) has a precise semantic meaning associated with it.
- This meaning should be taken into account by clients invoking the service, so that they can understand what functionality is offered by the service
- Semantics can be modeled:
  - using constraints (e.g., in case of domains having enumerable elements)
  - using ontologies (which formally define a vocabulary of terms and relationships)
  - using contracts (pre-conditions and post-conditions)
- In an integration scenario, the middleware infrastructure should preserve the semantics of the applications to be integrated as well as provide support for mediation (the transformation of messages between different representation by mapping concepts that are shared between all applications)
- If services are described with WSDL, there is very little semantics associated with them.
- Thus, there are many extensions to WSDL that can be used to model semantics, e.g., using the Resource Description Framework (RDF) and OWL (Web Ontology Language)



More information

- Take the EAI lecture, if you are interested in doing a big project using Web services
- Read the book:
- G. Alonso et al., **Web Services. Concepts, Architectures and Applications**, Springer, 2004
  - ISBN 3-540-44008-9
- ETH-BIB 783322
- ETH-INFK IK.04.1