



Information and
Communication Systems
Research Group



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

4 Universal Description, Discovery and Integration (UDDI)

Gustavo Alonso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
alonso@inf.ethz.ch
<http://www.iks.inf.ethz.ch/>



*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Background and historical perspective

Basic Problems to solve

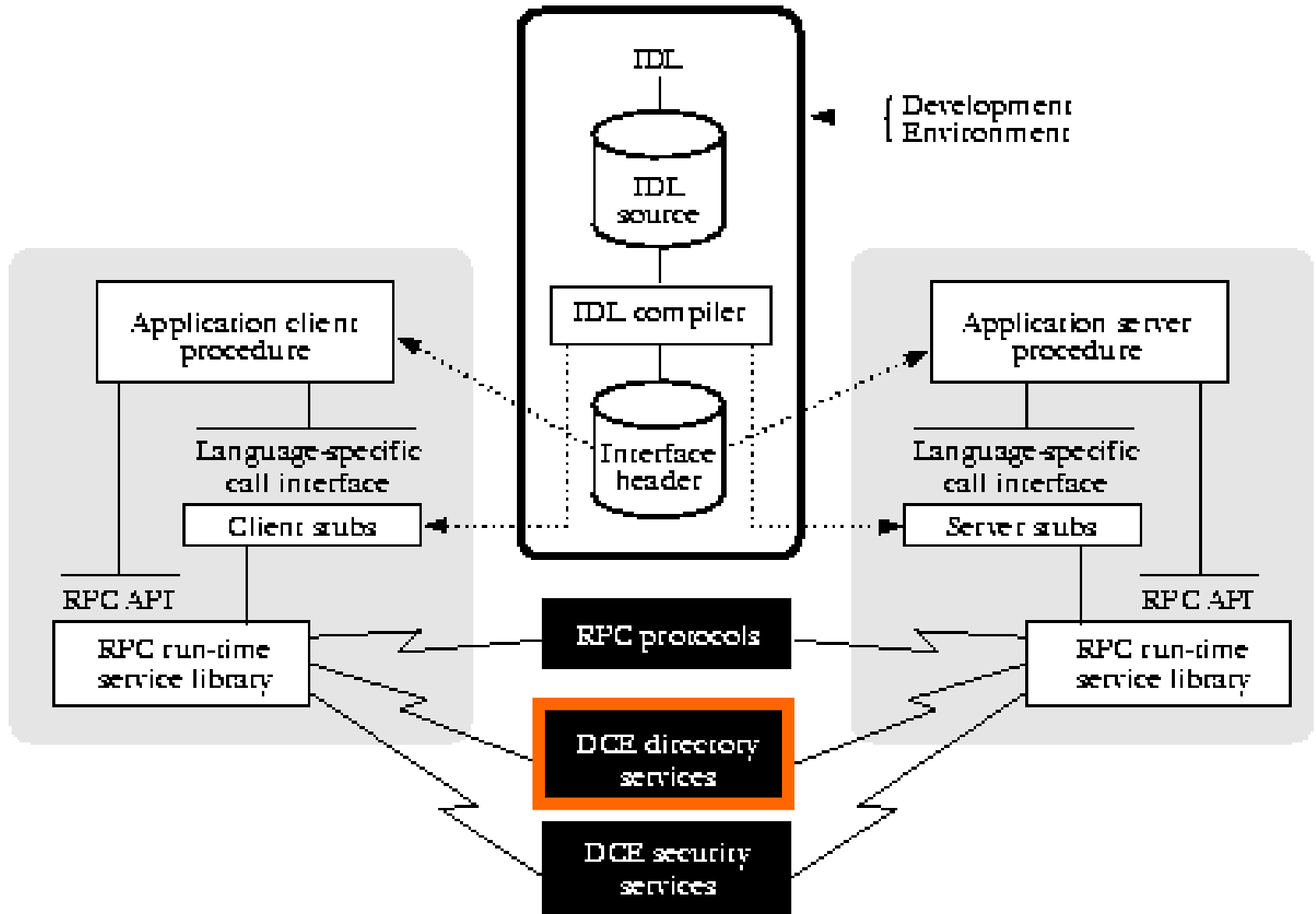
UDDI

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 then reconstructed)
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 ...

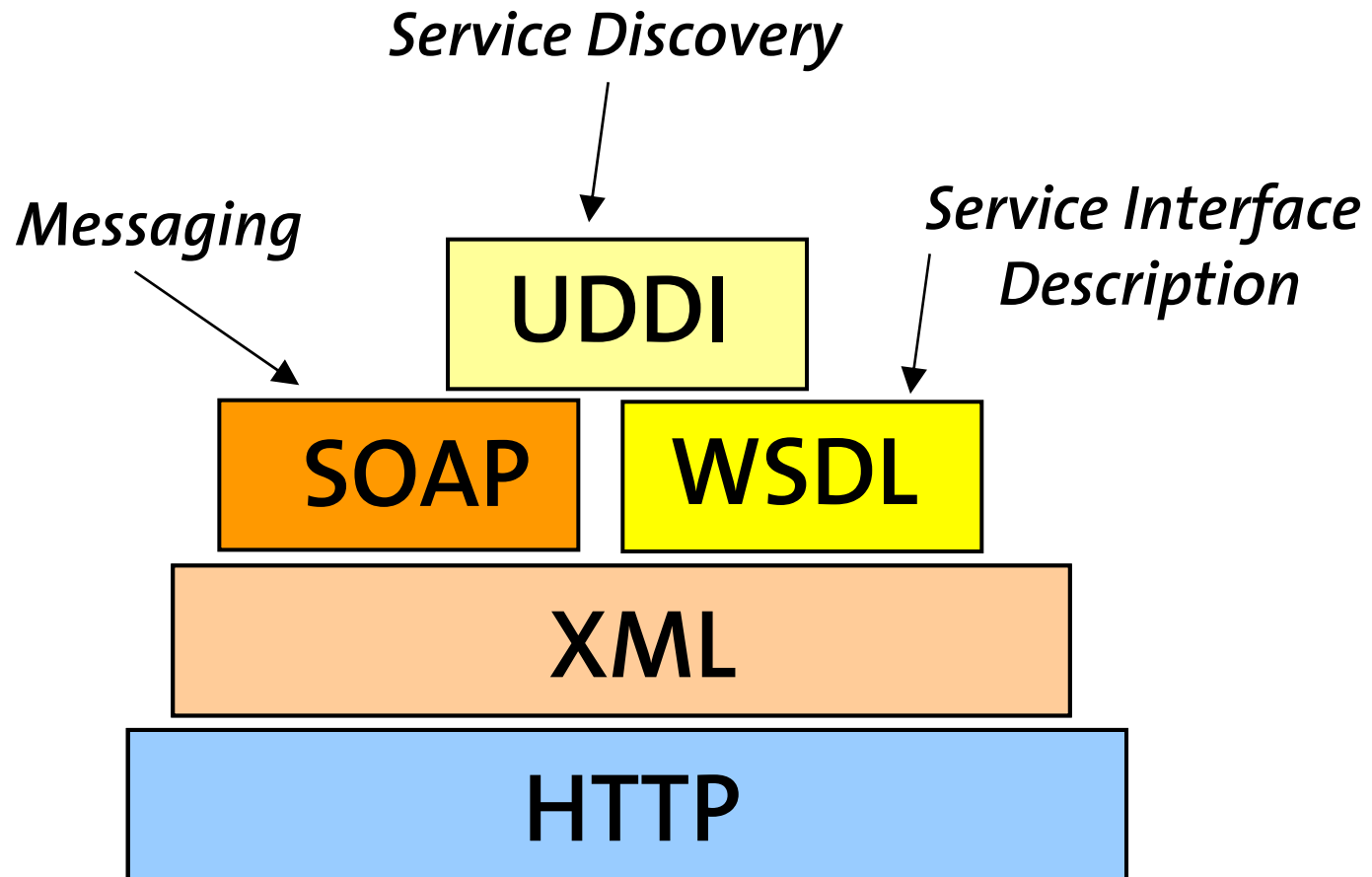
Binding in RPC

- A service is provided by a server located at a particular IP address and listening to a given port
- Binding is the process of mapping a service name to an address and port that can be used for communication purposes
- Binding can be done:
 - locally: the client must know the name (address) of the host of the server
 - distributed: there is a separate service (service location, name and directory services, etc.) in charge of mapping names and addresses. These service must be reachable to all participants
- With a distributed binder, several general operations are possible:
 - REGISTER (Exporting an interface): A server can register service names and the corresponding port
 - WITHDRAW: A server can withdraw a service
 - LookUP (Importing an interface): A client can ask the binder for the address and port of a given service
- There must also be a way to locate the binder (predefined location, environment variables, broadcasting to all nodes looking for the binder)

DCE architecture



Positioning UDDI

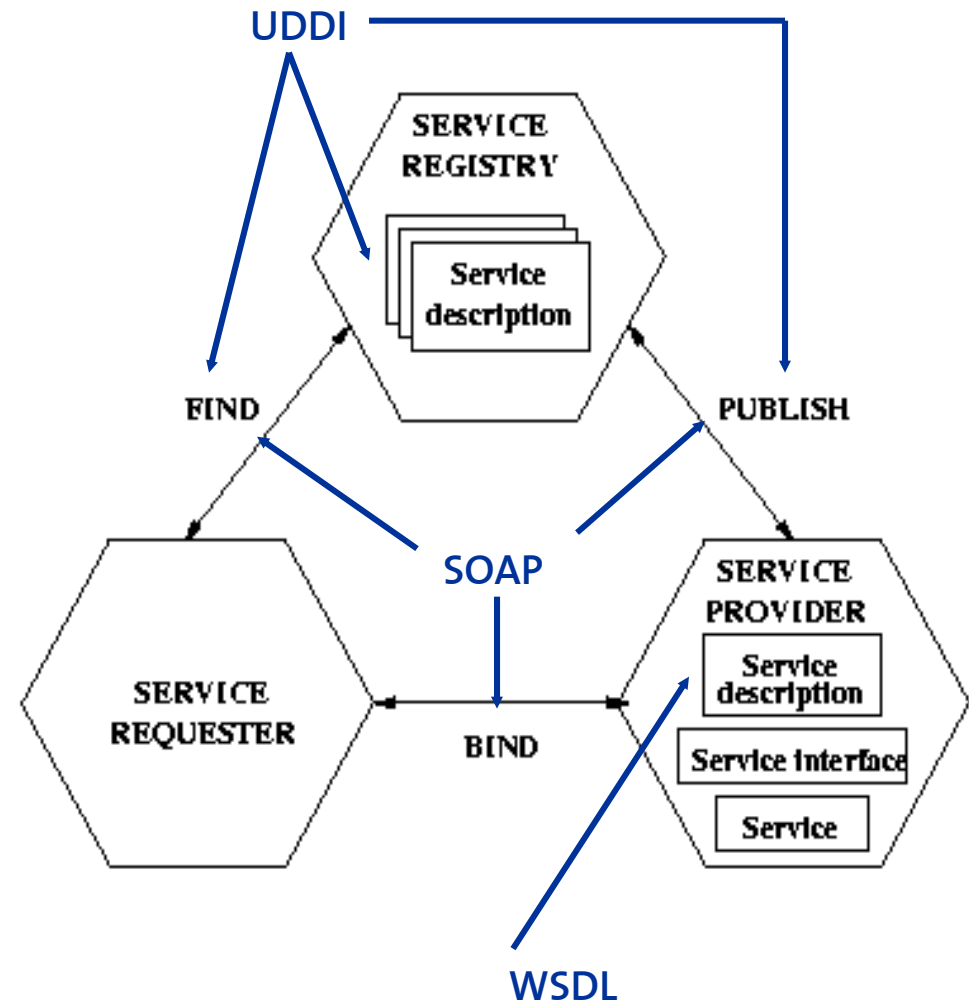


UDDI in the WS stack

Transport	HTTP, IIOP, SMTP, JMS		
Messaging	XML, SOAP	WS-Addressing	
Description	XML Schema, WSDL	WS-Policy, SSDL	
Discovery	UDDI	WS-MetadataExchange	
Choreography	WSCL	WSCl	WS-Coordination
Business Processes	WS-BPEL	BPML	WSCDL
Stateful Resources	WS-Resource Framework		
Transactions	WS-CAF	WS-Atomic Transactions WS-Business Activities	
Reliable Messaging	WS-Reliability	WS-ReliableMessaging	
Security	WS-Security SAML, XACML	WS-Trust, WS-Privacy WS-SecureConversation	
Event Notification	WS-Notification	WS-Eventing	
Management	WSDM	WS-Management	
Data Access	OGSA-DAI	SDO	

The role of UDDI

- Once it is possible to interact with any service provider using the standard SOAP protocol, it is still necessary to:
 - Describe the services (WSDL, Web Services Description Language)
 - **Discover the services** (UDDI, Universal Description, Discovery and Integration)





*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

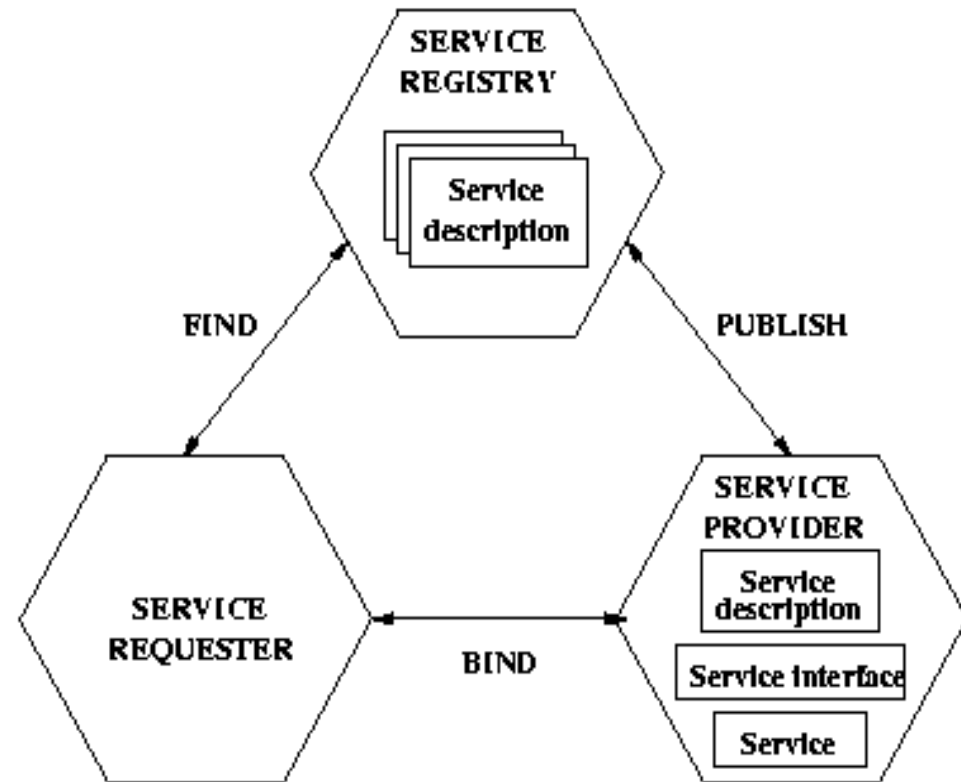
Universal Description, Discovery and Integration (UDDI)

What is UDDI?

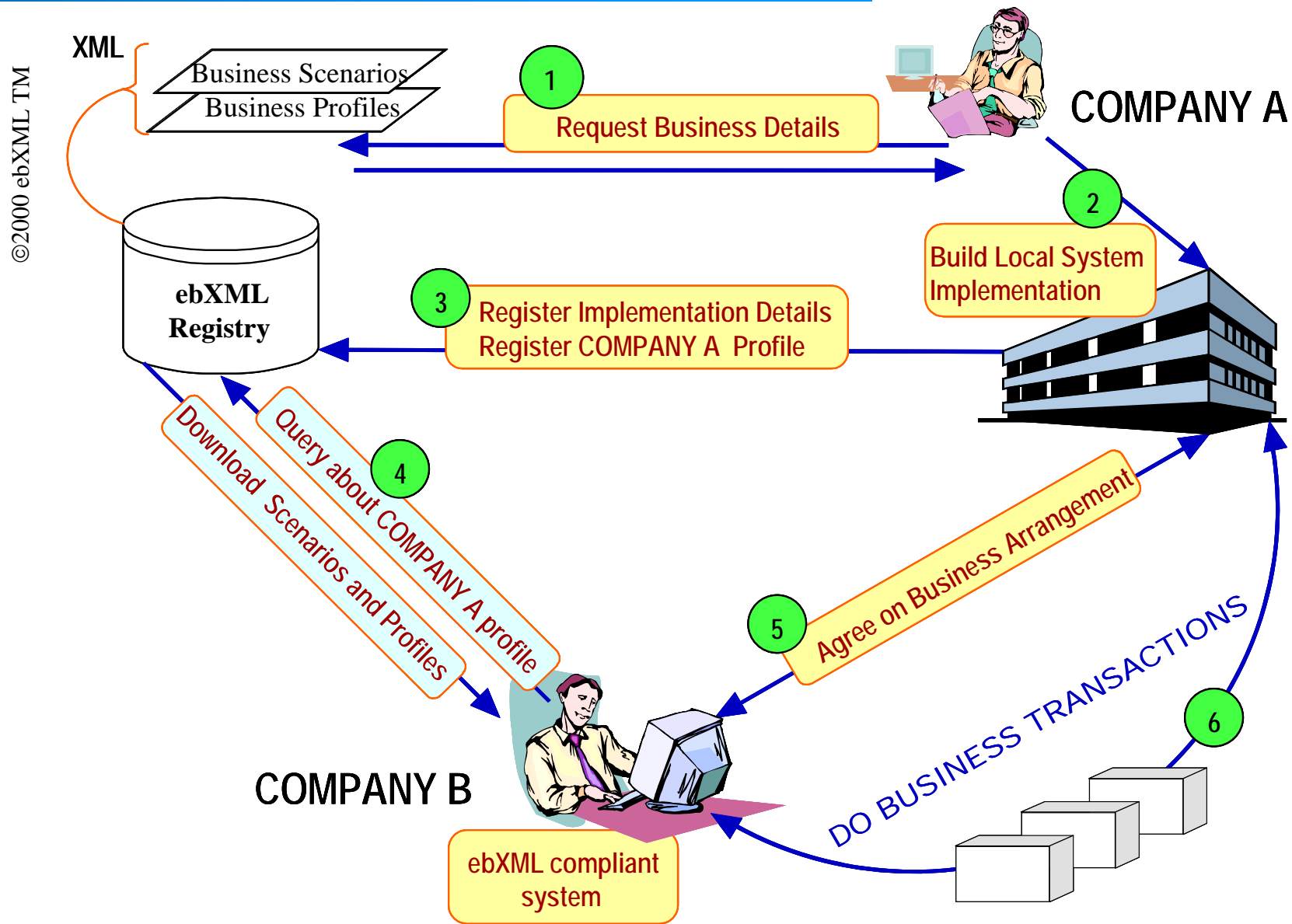
- The UDDI specification is probably the one that has evolved the most from all specifications we have seen so far. The latest version is version 3 (OASIS Standard Feb 2005):
 - version 1 defined the basis for a business service registry
 - version 2 adapted the working of the registry to SOAP and WSDL
 - 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 than 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 than 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



*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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 additional 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)

- 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

- The services provided by a business entity are described in business terms using `businessService` elements.
- A `businessEntity` can have several `businessServices` but a `businessService` belongs to one `businessEntity`
- The `businessService` can actually be provided by a different `businessEntity` than 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 a 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 tModels, 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 bindingTemplate 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

- A tModel is a generic container of information where designers can write any technical 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
 - “any concept that is not better represented by one of the other UDDI data structures”.
- 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





*Information and
Communication Systems
Research Group*



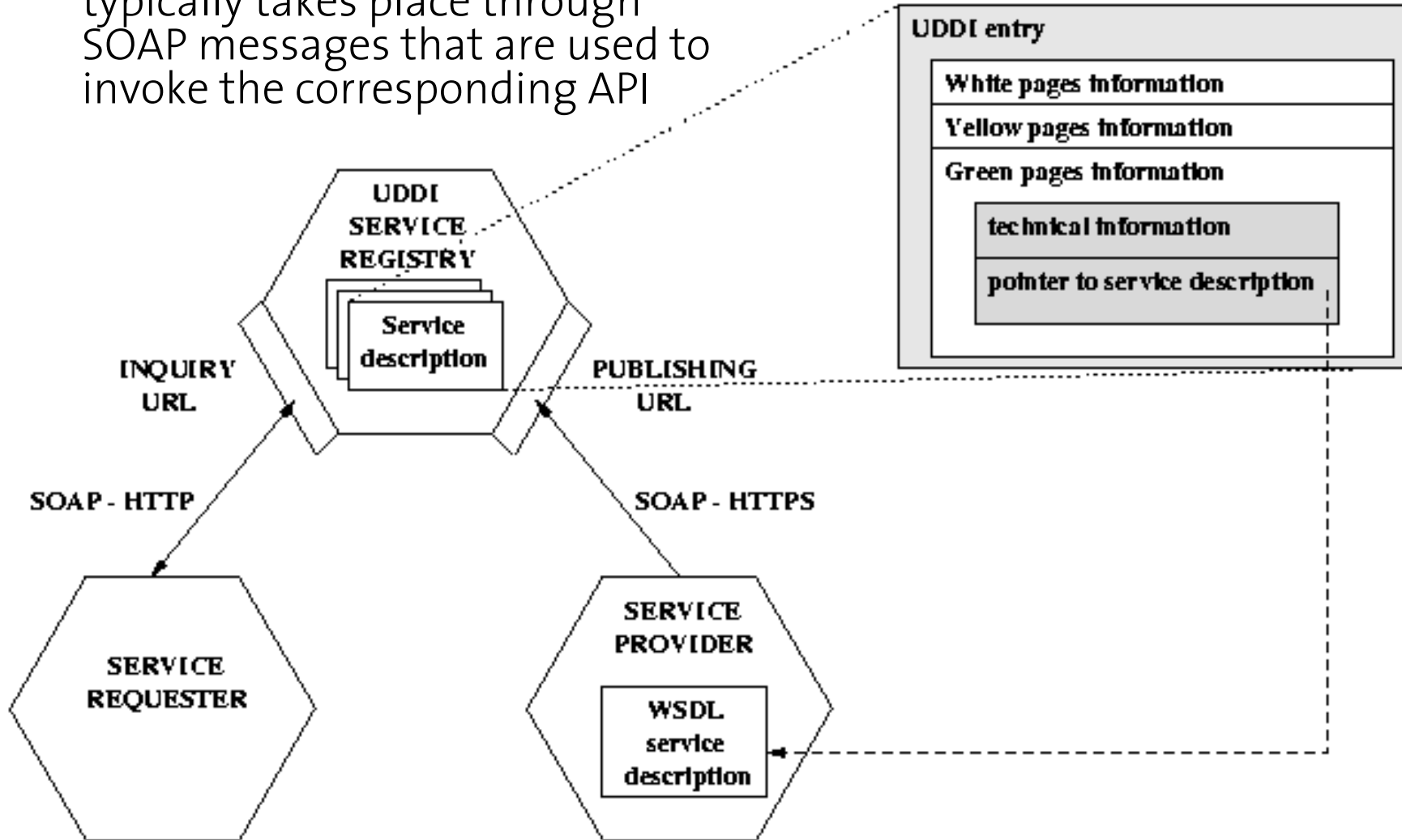
ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Interacting with a 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 UDDI 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 an 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

save_business
save_service
save_binding
save_tModel

Deletion

delete_business
delete_service
delete_binding
delete_tModel

Summary UDDI

- 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 constraints 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.



*Information and
Communication Systems
Research Group*



ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Limitations of UDDI

Hype and reality

- There were a few universal UDDI registries in operation (maintained by IBM, Microsoft, SAP, etc)
- These registries were very visible and often the first thing one saw of Web services
- Most of the entries in them did not work or did not correspond to any real service
- This has been a source of criticism to Web 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

UDDI Public Registries

- Former 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>

SAP

Homepage: <http://uddi.sap.com/>

Inquiry API :

<http://uddi.sap.com/uddi/api/inquiry>

Publish API :

<https://uddi.sap.com/uddi/api/publish>

Microsoft

Homepage: <http://uddi.microsoft.com/>

Inquiry API:

<http://uddi.microsoft.com/inquire>

Publish API :

<https://uddi.microsoft.com/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>

- The public UDDI Business registries provided by IBM, Microsoft and SAP have been discontinued since January 2006 (<http://uddi.microsoft.com/about/FAQshutdown.htm>)
- Since their launch in Sept. 2000, they accumulated over 50'000 service registration entries.

- The problem with UDDI is the initial ambitious goals:
 - the “Google” of web services (and standardized)
 - automatically find business partners worldwide
 - find the interface and build the application on the fly

- In reality:
 - Nobody does business with partners found at random in the Internet
 - Contract and SLAs are more than syntax, the legal aspect take precedence
 - Trust and knowing the partner are very important in practice
 - The interface describes the syntax, conversations and more complex functions are not yet sufficiently standardized
 - Most of the information in an UDDI repository is redundant if vertical standardization takes place

UDDI vs. directory services

- UDDI good ideas:
 - Services
 - Standardization
 - Different types of information to describe a service
- However, for SOA within a company, UDDI is not good enough:
 - No role base access
 - No life cycle management
 - No support for advanced features (governance)
- Products have started to deviate from the standard (IBM's WebSphere)