Nathanaël COTTIN

Web services

All and even more...

Nathanaël Cottin
www.ncottin.net

version 0.1.0 – 2010

Foreword

Nathanaël COTTIN

Widely used technology

Initiated by Ariba, IBM and Microsoft

Now supported and maintained by W3C, IETF and OASIS

Outline

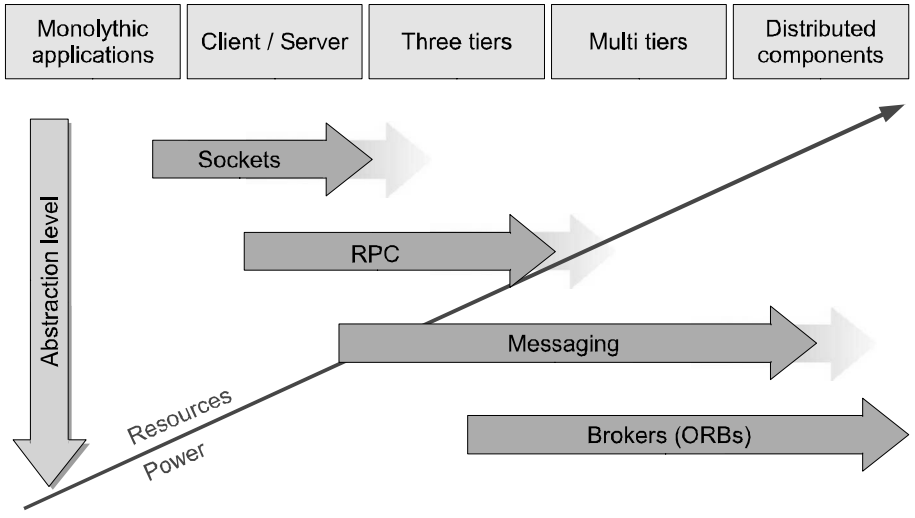
Nathanaël COTTIN

Nathanaël COTTIN

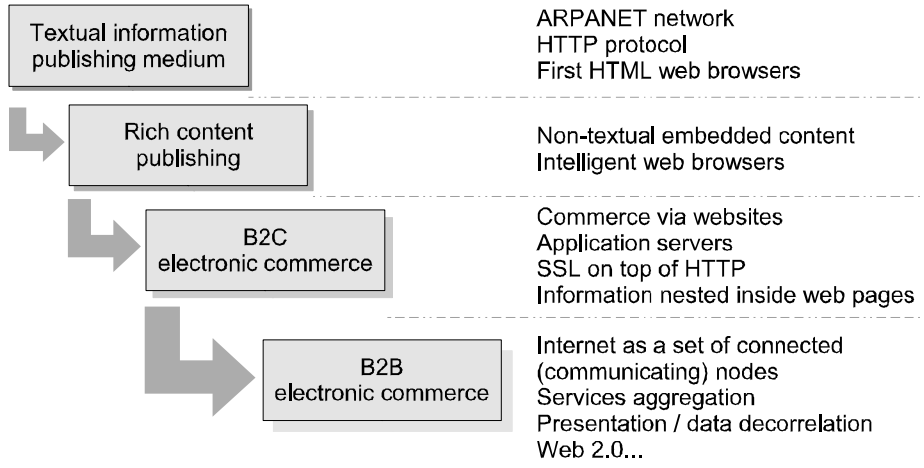
Introduction

Towards a new Internet web

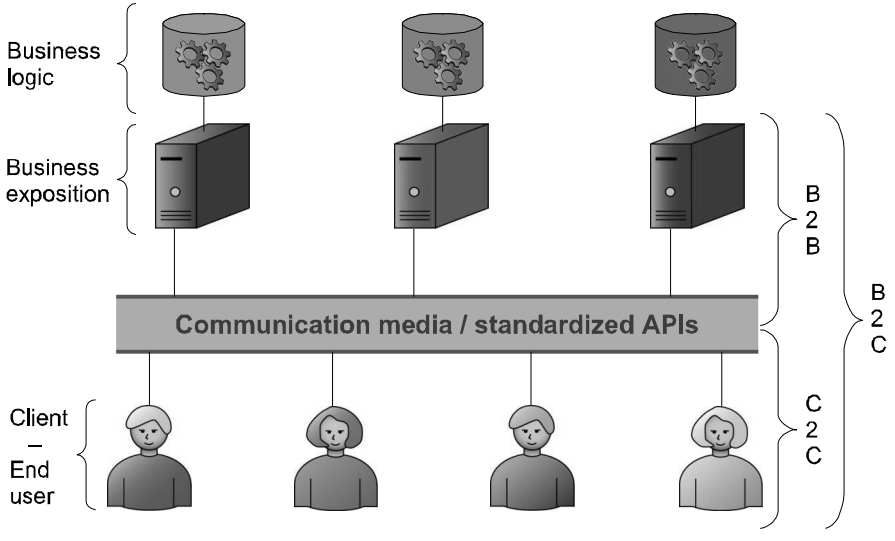
Information systems history



Brief evolution of the Internet



Distributed components over the Internet



Part 1

Web services concepts

- Terminology
- Web Services Architecture
- Key technologies

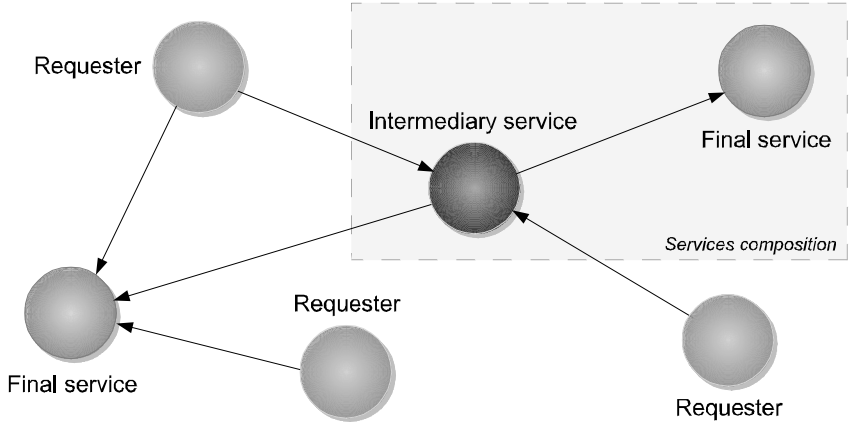
Web services as a distributed systems paradigm 1/6

Distributed system: set of discrete and autonomous entities (components) able to work together to provide requesters with complete or partial formatted information

Components are able to easily interact by means of a (standardized) communication medium called middleware

Web services as a distributed systems paradigm 2/6

Distributed system = (dynamic) oriented graph



Web services as a distributed systems paradigm 3/6

Distributed systems objectives:

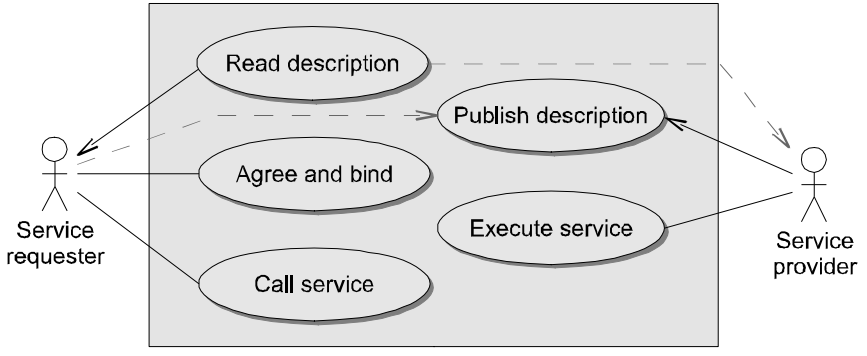
- **Efficiency**
- **Consistency**
- **Flexibility**
- **Robustness**

Used to define QoS criteria

Web services as a distributed systems paradigm 4/6

Distributed component:

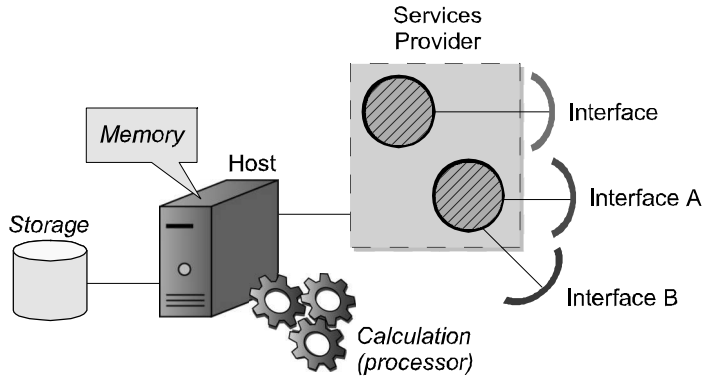
- Local or distant entity
- Composed of a set of execution tasks
- Publicly exposed by means of its interface(s)



Web services as a distributed systems paradigm 5/6

Host:

- Holds a services provider (SP) which realizes (local) services
- Provides memory, calculation and storage space



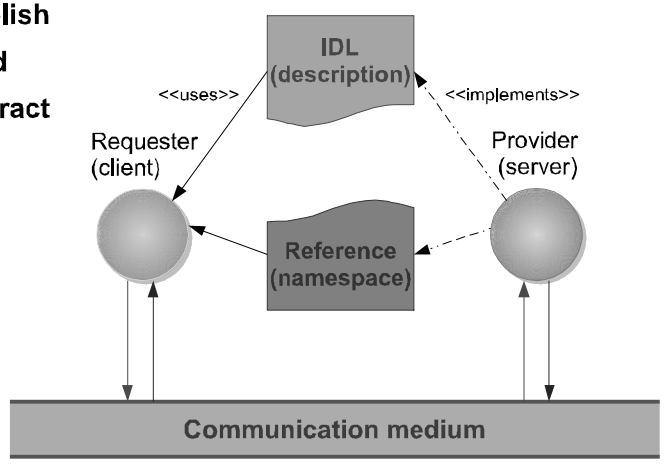
Web services as a distributed systems paradigm 6/6

Generic invocation process:

0. Publish

1. Find

2. Interact



Distributed systems challenges 1/2

- **Efficiency:**
  - Transport mechanics: latency, network unreliability
  - Transmitted data size
  - Requests management (scalability)
  - QoS guaranty
- **Consistency:**
  - Lack of shared memory, no global clock
  - Transactions handling
  - Failure scenarios support (compensation)
  - Interoperability
  - Updates incompatibilities

Distributed systems challenges 2/2

- **Flexibility:**
  - New services (components) integration
  - Automatic and dynamic composition
  - Migration, replication
- **Robustness:**
  - Disponibility
  - Persistence
  - Security: distributed authorizations

Service Oriented Architecture

SOA: type of distributed systems architecture

Properties:

- **Logical view**
- **Message-oriented**
- **Description-oriented**
- **Granularity**
- **Network-oriented**
- **Interoperable**
- **Platform-independent**

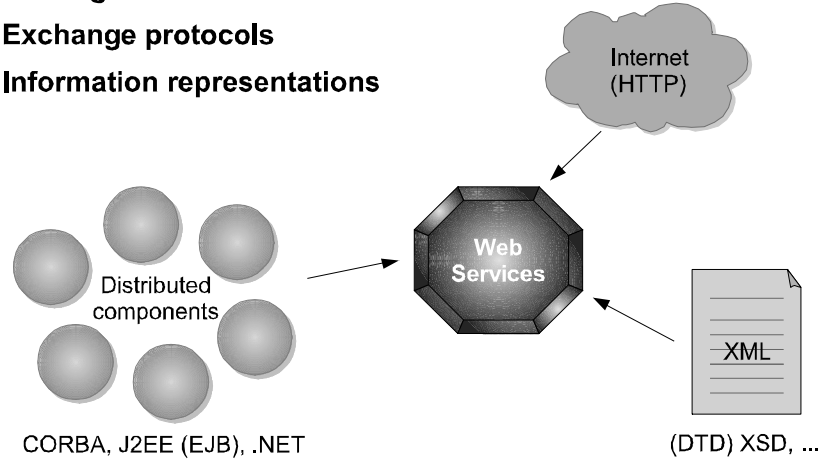
SOA and web services recommended usage

Most appropriate architectures for designing applications:

- **That must operate over the Internet**
- **Where reliability and performance (speed) cannot be guaranteed**
- **Where all requesters and providers do not need to be upgraded at once**
- **Where distributed components run on various platforms and vendor products**
- **Where an existing application needs to be exposed for use over a network and can be wrapped as a web service**

Web services key technologies

- **Paradigms**
- **Exchange protocols**
- **Information representations**



Web service and Agent definitions

Web service (as defined by W3C):

- Software system
- Designed to support interoperable machine-to-machine interaction over a network
- Has an interface described in a machine-processable format
- Provides some functionallity on behalf of its owner
- Other systems interact with the web service in a manner described by its descriptions

Agent:

- Concrete web service implementation
- Receives and sends messages
- Hosted by a Services Provider Agent

Web service description and semantics definitions

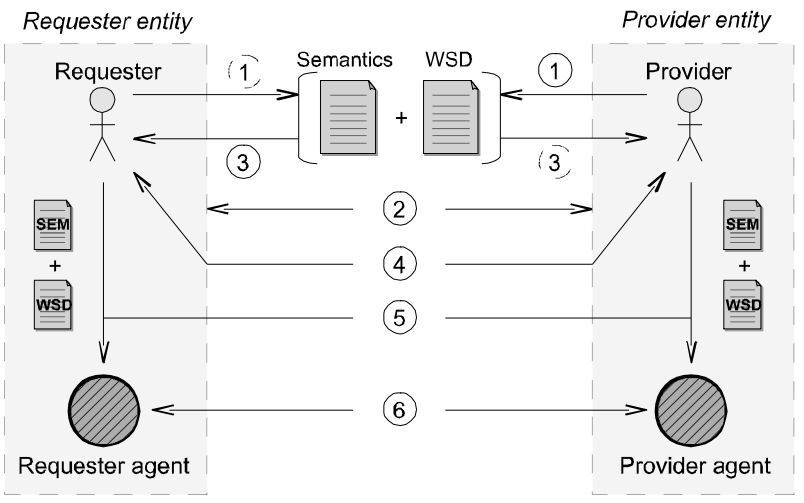
Web service description (WSD):

- Documents message exchange mechanics
- Machine-processable specification of the web service's interface
- Defines message formats, datatypes, transport protocols and transport serialization formats
- Specifies one or more network locations at which a provider agent can be invoked

Semantics:

- Legal or informal agreement between the requester and provider entities regarding the behaviour of a service (purpose and effects of the interaction to come)
- Not necessarily written or explicitly negotiated
- Machine-processable or human-understandable

Web service engagement: general process 1/2



Web service engagement: use of a discovery service 2/2

During second phase of the engagement process

Discovery:

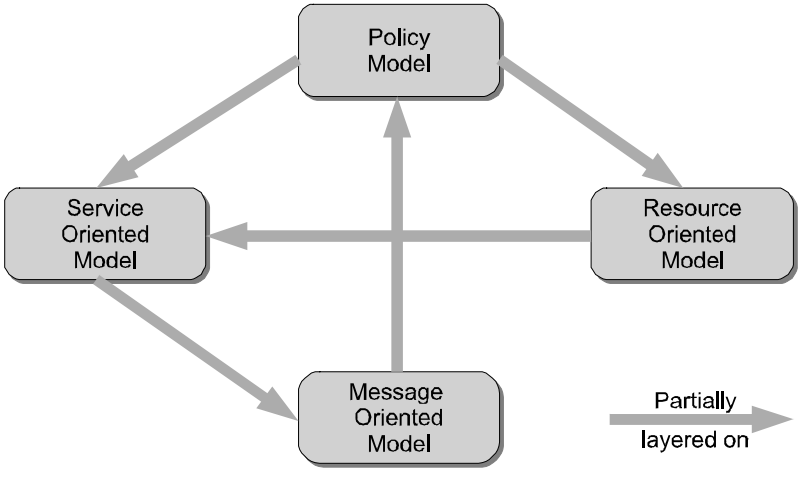
- Act of locating a machine-processable description of a service
- The located service must be appropriated (must meet a set of predefined criteria)

Discovery service:

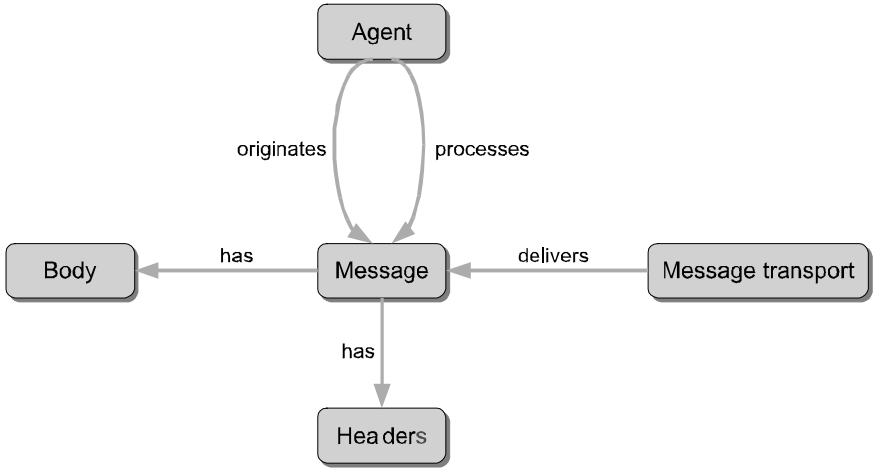
- Service which facilitates the process of performing discovery
- Manual or autonomous discovery processes

Functional description: machine-processable description of the functionalities (or partial semantics) of the service

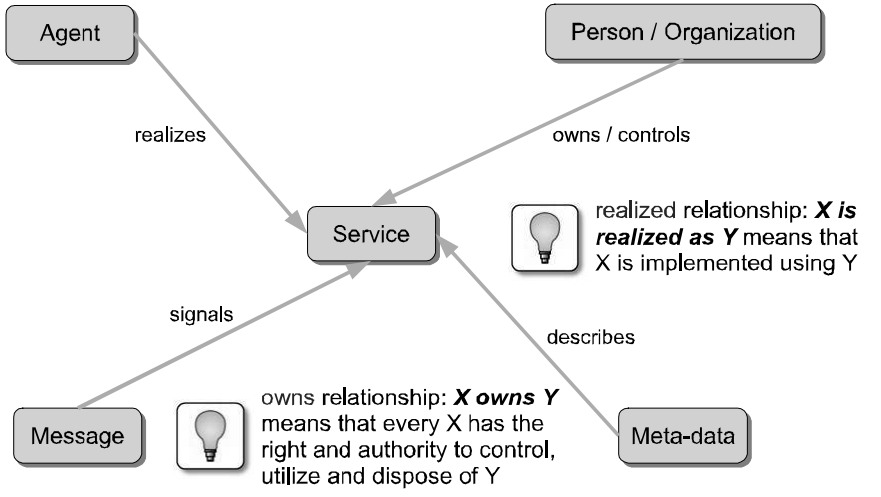
Web Services Architecture: meta model 1/6



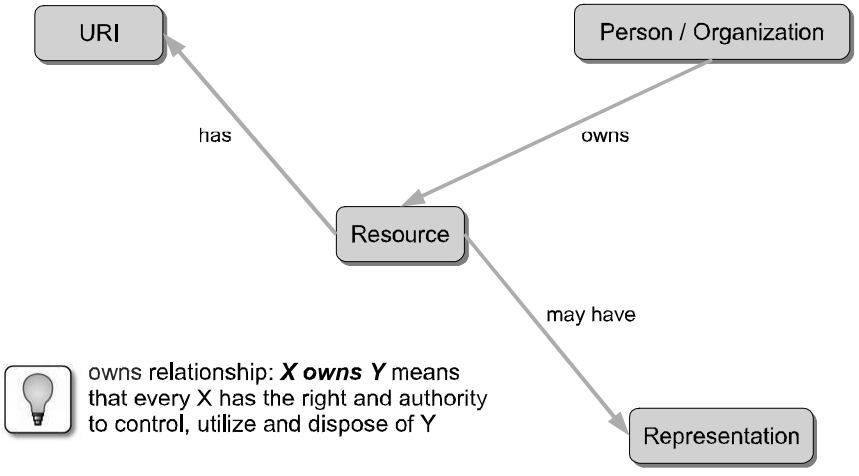
Web Services Architecture: MOM overview 2/6



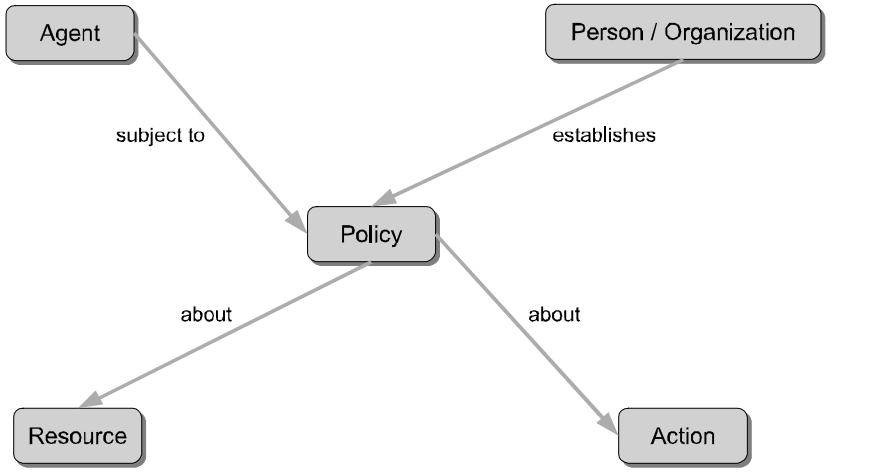
Web Services Architecture: SOM overview 3/6



Web Services Architecture: ROM overview 4/6

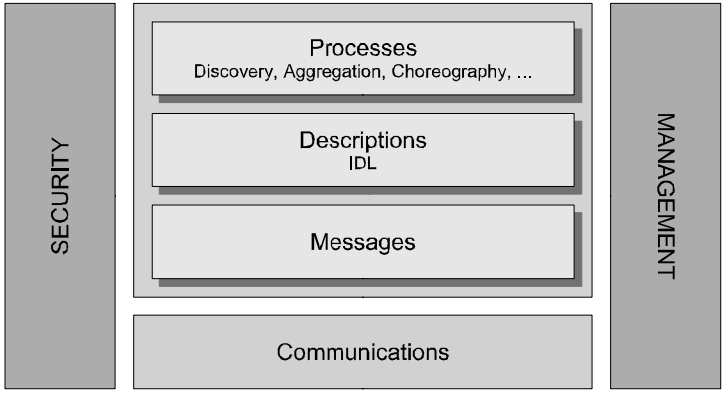


Web Services Architecture: PM overview 5/6



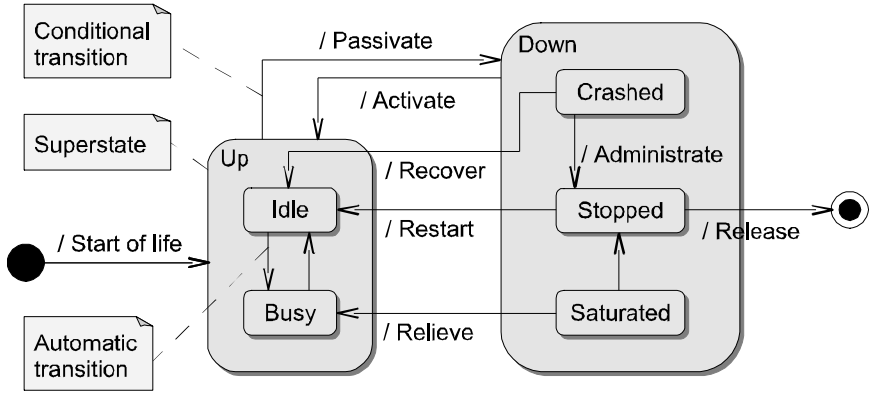
Web Services Architecture: architecture stack 6/6

- **Based on distributed systems implementations backing**
- **Intended to describe minimal web services characteristics**



Web service lifecycle

Service realization by means of an SPA:



Web services classification

- **Business information**: give information to consumers (including other services)
- **Business integration**: transactional aggregations and compositions of business web services
- **Business process externalization**: external web services cunjunction to compute local (own) and tiers information

Web services standards

Communication layer:

- **HTTP**: HyperText Transfer Protocol
- **XML (including SOAP)**: eXtended Markup Language

Publication and discovery:

- **WSDL**: Web Services Description Language
- **WSIL**: Web Services Inspection Language
- **UDDI**: Universal Description Discovery and Integration

Work-in-progress:

- OASIS WS-*galaxy
- Private initiatives

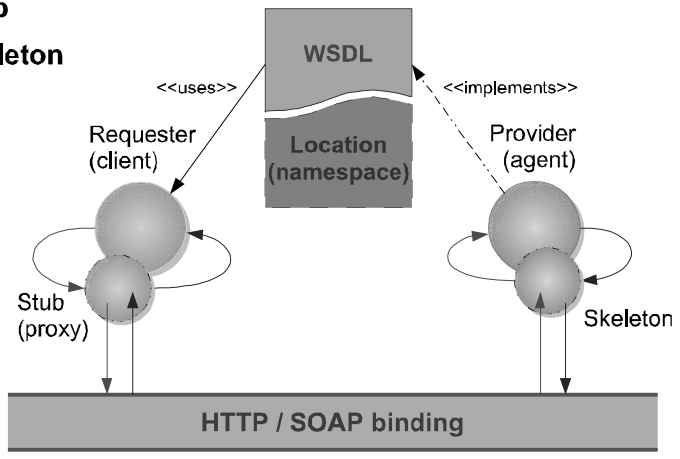
Web services properties

- **Self-containance**: based on HTTP and XML formats
- **Self-description**: information transmitted along with format description
- **Modularity**: services aggregation using composition
- **Interoperability**: well-known and standardized technologies
- **Platform independence**: textual exchanges
- **Persistence**: handled by web services container

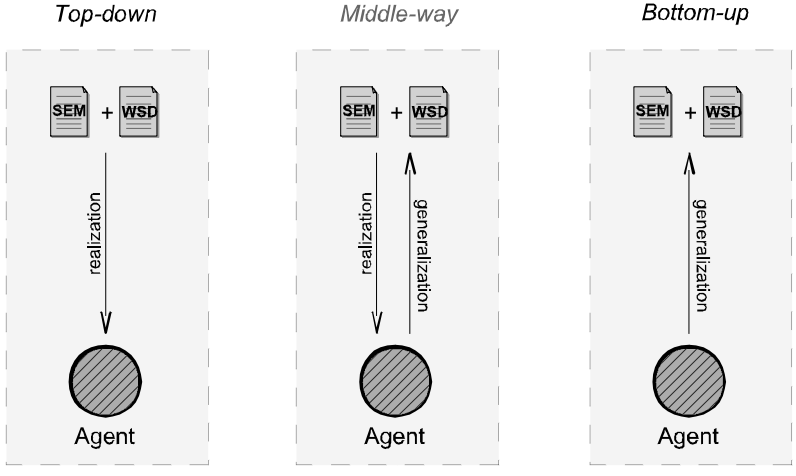
Web service invocation detailed process

Transparency:

- **Stub**
- **Skeleton**



SOAs implementations approaches



Part 2

Core technologies

- HTTP
- XML technologies (DTD, Infoset, Schema & Namespace)

HTTP brief history

HTTP(S): (Secure) HyperText Transfer Protocol

- 1990 – 1996: **HTTP 0.9**
  - No meta-data
  - Response directly sent back to requester
  - No cache
- 1997: **HTTP 1.0**
  - MIME-based headers and meta-data
  - Simple caching ability
  - `HEAD` and `POST` methods
- 1999: **HTTP 1.1**
  - Smart caching
  - Content negociation
  - Connection persistence
  - Chunking
- 1995: **SSL 2.0**
- 1996: **SSL 3.0**
- 1999: **TLS 1.0**

HTTP: communication layer core standard

CREATE	RETRIEVE	UPDATE	DELETE	MISC
PUT	HEAD	**POST**	DELETE	CONNECT
(POST)	**GET**	(PUT)		TRACK
	OPTIONS			
	TRACE			
	(POST)			

HTTP examples: GET request / response 1/3


```
GET /misc.php HTTP/1.1
Host: www.ncottin.net
Referer: http://www.ncottin.net:80/
User-Agent: Mozilla/5.0 ... Firefox/2.0.0.6
Connection: keep-alive
```

```
HTTP/1.1 200 OK
Server: Apache/2.2.4 (Win32) PHP/5.2.0
X-Powered-By: PHP/5.2.0
Content-Length: 5178
Content-Type: text/html

<html>
...
</html>
```


HTTP examples: URL-encoded POST request 2/3


```
POST /misc.php HTTP/1.1
Host: www.ncottin.net
Referer: http://www.ncottin.net:80/
Accept: text/xml,text/html;q=0.9,image/png,*/*;q=0.5
User-Agent: Mozilla/5.0 ... Firefox/2.0.0.6
Connection: keep-alive
Content-Length: 28

arg1=val1&arg2=val21%20val22
```

HTTP examples: mime-encoded POST request 3/3


```
POST / HTTP/1.1
Host: www.ncottin.net
Referer: http://www.ncottin.net:80/
Content-Type: multipart/form-data; boundary=...
Content-Length: 183

--[boundary]
Content-Disposition: form-data; name="arg1"

val1
--[boundary]
Content-Disposition: form-data; name="arg2"

val21 val22
--[boundary]--
```


HTTP common header fields 1/2

Accept: defines accepted media types for the response. All media types if not present

Accept-Charset: acceptable character sets (UTF-8, ISO-8859-1, ...)

Accept-Encoding: restricts the content coding acceptable for the response

Accept-Language: indicates preferred language for the response

Content-Type: media type of the entity body sent to the requester

HTTP common header fields 2/2

Host: specifies Internet host and port of the requested resource

Referer: absolute URI of the ressource which leaded to the requested URI

Cache-Control: specifies directive which must be followed by caching mechanisms along the request/response chain

Content-Length: size of the entity body, expressed in decimal number of octets (≥ 0)

Date: date and time of the message

Cookie: cookie value

HTTP 1.1 main response status codes

STATUS CODE	TYPE	DESCRIPTION
200	Success	OK
400	Client error	BAD REQUEST
401	Client error	UNAUTHORIZED
403	Client error	FORBIDDEN
404	Client error	NOT FOUND
406	Client error	NOT ACCEPTABLE
408	Client error	REQUEST TIMEOUT
500	Server error	INTERNAL SERVER ERROR

XML brief history

- 1986 – 1988: **SGML**
  Standard Generalized Markup Language
- 1990 – 1999: **HTML 1.0 – 4.0**
  HyperText Markup Language
- 1998 – 2006: **XML 1.0 – 1.1** **(XHTML)**
  eXtensible Markup Language
  - Well-formedness
  - Validity
- 2001: **canonical XML 1.0**

XML: basic example


```
<my-element attribute1="value1" attribute2="value2">

   <empty-sub-element/>
   <sub-element>
      Sub element value
   </sub-element>
   <sub-element>
      <![CDATA[<h1>An HTML level-1 head</h1>]]>
   </sub-element>
</my-element>
```

XML documents automatic validation (conformance)?

XML Document Type Definition example


```
<!-- sample.dtd: defines "my-element" element -->
<!ELEMENT my-element (empty-sub-element? sub-element+)>
<!ATTLIST my-element elementAttr>

<!ATTLIST elementAttr
   attribute1 CDATA #REQUIRED
   attribute2 CDATA #FIXED "value2"
   attribute3 (false|true|0|1) "true">

<!ELEMENT empty-sub-element EMPTY>

<!ELEMENT sub-element (#PCDATA)>
```


XML Schema: definition

The purpose of a schema is to define a class of XML documents, called XML instance documents

An XML instance document is an XML document which conforms to a particular schema

The main objective is to provide automatic validation of XML instance documents in terms of DTD as well as elements and attributes values (i.e. types and occurrences and values restrictions)

XML Schema: example


```
<!-- sample.xsd: defines "my-element" element -->
<xsd:complexType name="my-element">
   <xsd:sequence>
      <xsd:element name="empty-sub-element" minOccur="0">
         <xsd:complexContent/>
      </xsd:element>
      <xsd:element name="sub-element" type="xsd:string"
       minOccur="1" maxOccur="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="attribute1" type="xsd:string"
     use="required"/>
   <xsd:attribute name="attribute2" type="xsd:string"
     fixed="value2"/>
   <xsd:attribute name="attribute3" type="xsd:boolean"
     use="optional" default="true"/>
</xsd:complexType>
```


XML Schema: annotations

Used to document schemas for humans and applications

```
<xsd:element name="somethingToSay">
   <xsd:annotation>
      <xsd:documentation xml:lang="en">
         This is an annotation
      </xsd:documentation>
   </xsd:annotation>
   ...
</xsd:element>
```


XML Schema: built-in simple types (excerpt) 1/3

SIMPLE TYPE	EXAMPLES
base64Binary	BAEdQ/wMEREFU+S==
boolean	false, true, 0, 1
byte	-127, ..., 127
date	2007-09-19, 1999-12-31
decimal	-12.34, 0, 1.23, 99.00
double	-INF, 0, 2.83E-4, 48, INF, NaN
hexBinary	0FB7

XML Schema: built-in simple types (excerpt) 2/3

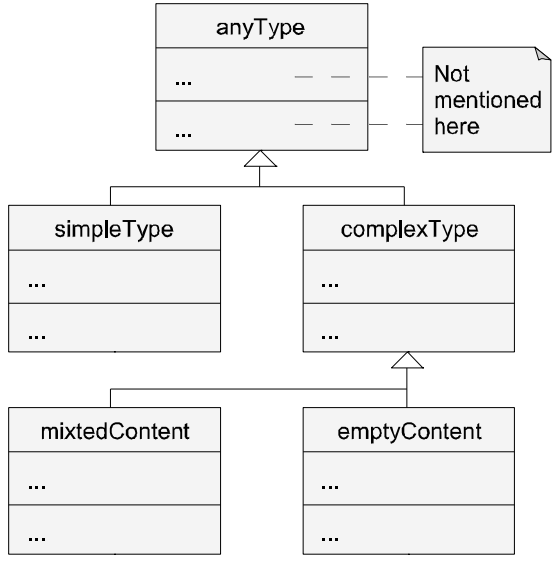
SIMPLE TYPE	EXAMPLES
int	-2147483648, ..., 2147483647
integer	..., -1, 0, 1, ...
long	-9223372036854775808, ..., 9223372036854775807
negativeInteger	..., -2, -1
nonNegativeInteger	0, 1, 2, ...
nonPositiveInteger	..., -1, 0
positiveInteger	1, 2, ...

XML Schema: built-in simple types (excerpt) 3/3

SIMPLE TYPE	EXAMPLES
short	-32768, ..., 32767
string	This is a string
time	13:26:48, 13:26:48.000-05:00
unsignedByte	0, 123
unsignedInt	0, ..., 4294967295
unsignedLong	0, ..., 18446744073709551615
unsignedShort	0, ..., 65535

XML Schema: elements types

- **Simple types**
- **Complex types**
- **Mixted content**
- **Empty content**
- **`anyType` type**



XML "element" schema description


```
<element
        abstract = boolean : false
        block = (#all | List of (extension | restriction |
          substitution)
        default = string
        final = (#all | List of (extension | restriction)
        fixed = string
        form = (qualified | unqualified)
        id = ID
        maxOccurs = (nonNegativeInteger | unbounded) : 1
        minOccurs = nonNegativeInteger : 1
        name = NCName
        nillable = boolean : false
        ref = QName
        type = QName>
    Content: (annotation?, (simpleType | complexType)?,
      (unique | key | keyref)*)
</element>
```


XML "attribute" schema description


```
<attribute
        default = string
        fixed = string
        form = (qualified | unqualified)
        id = ID
        name = NCName
        ref = QName
        type = QName
        use = (optional | prohibited | required) : optional>
    Content: (annotation?, simpleType?)
</attribute>
```

XML Schema: occurrences constraints 1/2

Elements:

- **Minimum:** `minOccurs`
- **Maximum:** `maxOccurs`
- **`use` attribute**

Attributes:

- **Minimum:** `0`
- **Maximum:** `1`

XML Schema: facets constraints 2/2

Properties used to restrict existing types

- **length**
- **minLength**
- **maxLength**
- **pattern**
- **enumeration**
- **whiteSpace**
- **minInclusive**
- **maxInclusive**
- **minExclusive**
- **maxExclusive**
- **totalDigits**
- **fractionDigits**

XML Schema extra types: derived types 1/2

Create new types from existing (built-in or derived) types

Example: percentage

```
<xsd:simpleType name="percentage">
   <xsd:restriction base="xsd:integer">
      <xsd:minInclusive value="0"/>
      <xsd:maxInclusive value="100"/>
   </xsd:restriction>
</xsd:simpleType>
```


XML Schema extra types: anonymous types 2/2

Define in-line types within the XSD

Example: in-line percentage element

```
<xsd:element name="eltName">
   <xsd:simpleType>
      <xsd:restriction base="xsd:integer">
         <xsd:minInclusive value="0"/>
         <xsd:maxInclusive value="100"/>
      </xsd:restriction>
   </xsd:simpleType>
</xsd:element>
```

XML Schema: regular expressions 1/2

Defined using `pattern` facet

EXP	MATCH	EXAMPLES
.	Any character	A, Z, a, z, 4, -
\s	Whitespace	<SPACE>, <TAB>
\d	Digit character	0, 1, 6, 9
\D	Non-digit character	a, A, ., -, *
\w	Word character (letter | digit)	a, A, z, Z, 0, 9
|	Alternative	a|z|0|9
[ ]	Alternatives	[az09]
-	Range	[a-z0-9]
^	Exclusion	^1, ^0-9
\	Protection character	\+, [\-a-z], [0-9\[]

XML Schema: regular expressions 2/2

Occurrences representation:

EXP	OCCURRENCES	REGEX	MATCHES
?	0 or 1	a?b	b, ab
*	0, 1 or more	a*b	b, ab, aaaab
+	1 or more	a+b	ab, aaaab
{X}	X occurrences	a{2}b	aab
{X,Y}	Between X and Y (inclusive)	a{2,3}b	aab, aaab
{X,}	X or more	a{2,}b	aab, aaaaab

Parenthesis used to define blocks:

- **(ab)*\+c → +c, ab+c, abab+c, ababab+c**
- **(a|b)(c|d)? → a, b, ac, ad, bc, bd**

Namespaces: overview 1/4

Use of different vocabularies

XML namespace:

- Method for qualifying elements and attributes used in XML documents
- Associates XML elements and attributes names with an URI

- **Modularity**
- **Reusability**
- **Based on URIs**

Namespaces: example 2/4


```
<?xml version="1.1"?>
<html:html xmlns:html="http://www.w3.org/1999/xhtml">
   <html:head>
      <html:title>Informational web page</html:title>
   </html:head>
   <html:body>
      <html:p>Please visit
         <html:a href="http://www.ncottin.net">my
          website</html:a> for other courses
      </html:p>
   </html:body>
</html:html>
```

Namespaces: locals qualification 3/4

- **Unqualified names:**

```
<schema xmlns="http://www.w3.org/2001/XMLSchema">
   ...
   <element name="elem" type="string"/>
   ...
</schema>
```

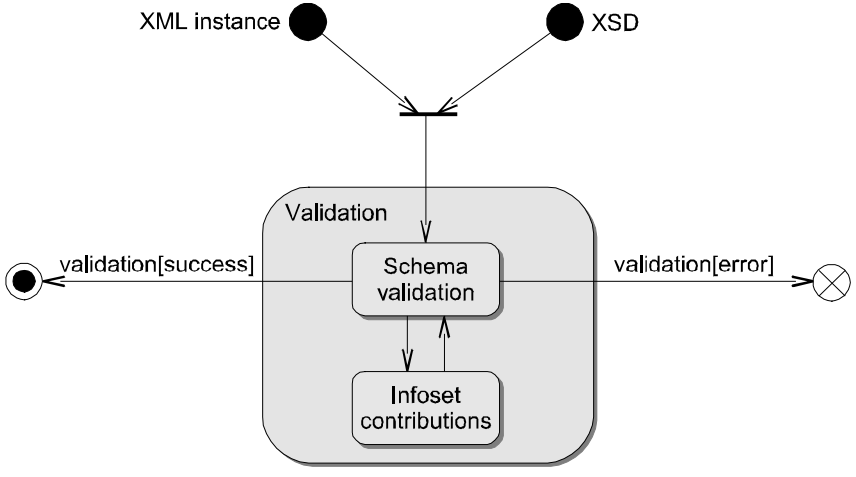
- **Qualified names (explicit or implicit):**
  - Prefixed names
  - Unprefixed names

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:qual="http://example.ncottin.net">
   ...
   <element name="elem" type="qual:myType"/>
   ...
</schema>
```

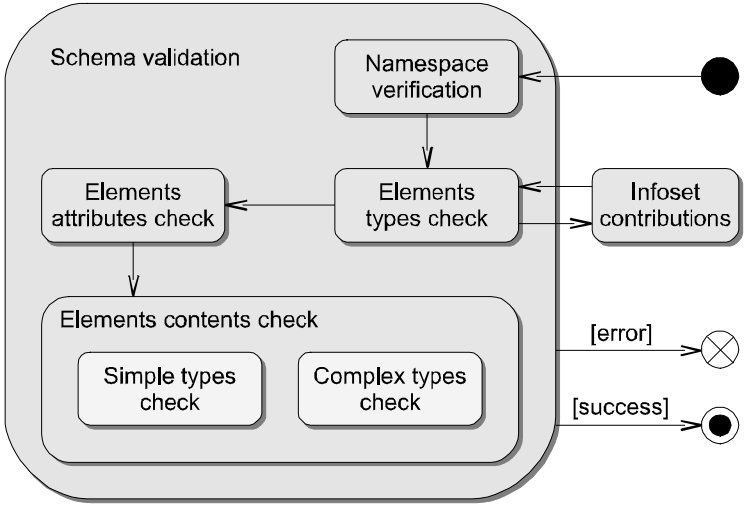

Namespaces: applying namespaces 4/4

- **Scope**
- **Multiple namespaces**
- **Default namespace**
- **Target namespace**

XML conformance verification against XML Schema 1/2



XML conformance verification against XML Schema 2/2



XML Infoset

XML Infoset: describes an XML document with expanded entity references expressed as a Document Information Item

Document Information Item properties:

- **XML version**
- **Base URI of the document entity**
- **List of Element Information Items**
- **...**

Expressed using an RDF Schema

RDF Schema

RDF Schema: general-purpose language for representing information in the Internet web

RDF represents:

- **Resources**
- **Relationships between resources**

W3C specifies an RDF representation for the XML Infoset

Part 3

Common bindings descriptions

- REST
- XML-RPC
- SOAP, SOAP-RPC
- Other bindings: WDDX, JSON

REST binding: request / response example

REST allows manipulating computable information using HTTP requests.

REST is only an evolution of the Internet

REST GET request example:

```
GET /hello/sayHelloToIdentity/?first=Nathanael&last=COTTIN
Host: www.ncottin.net
...
```

Response example:

```
<?xml version="1.0"?>
<resp status="OK">
   <value>Hello Nathanael COTTIN!</value>
</resp>
```

REST binding: getting parts 1/2

Getting REST parts:

```
GET /purchaseService/items
Host: www.ncottin.net
...
```

REST response:

```
<?xml version="1.0"?>
<i:Items xmlns:i="http://www.ncottin.net/purchaseService"
         xmlns:xlink="http://www.w3.org/1999/xlink">
   <Item id="0001"
         xlink:href="http://www.ncottin.net/items/0001"/>
   <Item id="0002"
         xlink:href="http://www.ncottin.net/items/0002"/>
</i:Items>
```


REST binding: using parts 2/2

Getting a REST part details:

```
GET /items/0001/
Host: www.ncottin.net
...
```

Returned part details:

```
<?xml version="1.0"?>
<i:Item xmlns:i="http://www.ncottin.net/purchaseService"
        xmlns:xlink="http://www.w3.org/1999/xlink">
   <Item-ID>0001</Item-ID>
   <Description>Part description here</Description>
   <ItemDetails xlink:href="details"/>
   ...
</i:Item>
```


XML-RPC binding: request example


```
<?xml version="1.0"?>
<methodCall>
   <methodName>hello.sayHelloToIdentity</methodName>
   <params>
      <param>
         <value>Nathanael</value>
      </param>
      <param>
         <value>COTTIN</value>
      </param>
   </params>
</methodCall>
```

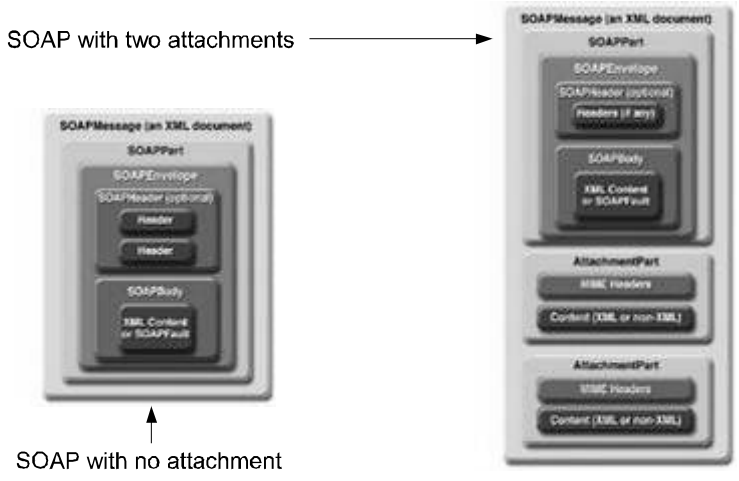

XML-RPC binding: response example


```
<?xml version="1.0"?>
<methodResponse>
   <methodName>hello.sayHelloToIdentity</methodName>
   <params>
      <param>
         <value>Hello Nathanael COTTIN!</value>
      </param>
   </params>
</methodResponse>
```

SOAP: an Internet revolution

- **XML-RPC** **1995**
  - Mechanism for invoking distant procedures
  - Stateless invocations
- **SOAP 1.0** **11/1999**
  - IBM, Userland Software, DevelopMentor, Apache group, Microsoft Corp.
  - XML-RPC adapted to object-oriented programming
- **SOAP 1.1** **05/2000**
  - Protocol independence: "binding"
  - RPC invocations as a possible application of SOAP
  - SOAP 1.1 with attachments **12/2000**
- **SOAP 1.2** **04/2007**

SOAP 1.1 binding: SOAP with attachments overall format



SOAP 1.1 binding: SOAP with attachments example 1/3


```
POST /mySOAPWebService HTTP/1.1
Host: www.ncottin.net
Content-Type: Multipart/Related; boundary=MIME_boundary;
  type=text/xml; start="mainID"
Content-Length: ...
SOAPAction: http://schemas.ncottin.net/SOAPAttach/sample
...

--MIME_boundary
Content-Type: text/xml; charset=UTF-8
Content-Transfer-Encoding: 8bit
Content-ID: mainID
```

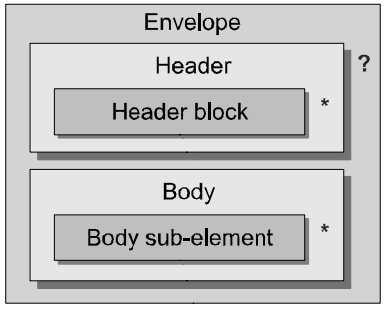

SOAP 1.1 binding: SOAP with attachments example 2/3


```
<?xml version="1.0"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/">
   <env:Header/>
   <env:Body>
      <att:attachments id="mainID"
        xmlns:att="http://schemas.ncottin.net/SOAPAttach/
        sample">
         <attach href="cid:attachment1ID"/>
         <attach href="cid:attachment2ID"/>
      </att:attachments>
   </env:Body>
</env:Envelope>
```

SOAP 1.1 binding: SOAP with attachments example 3/3


```
--MIME_boundary
Content-Type: image/gif
Content-Transfer-Encoding: base64
Content-ID: attachment1ID

...Base64 encoded GIF image data...
--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: attachment2ID

...Raw JPEG image data...
--MIME_boundary--
```


SOAP 1.2 binding: overall format



SOAP 1.2 binding: request example


```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
  envelope">
   <env:Header>
   ...
   </env:Header>
   <env:Body>
      <h:sayHelloToIdentity
            xmlns:h="http://soap.ncottin.net/hello">
         <h:first>Nathanael</h:first>
         <h:last>COTTIN</h:last>
      </h:sayHelloToIdentity>
   </env:Body>
</env:Envelope>
```


SOAP 1.2 binding: response example


```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
  envelope">
   <env:Header>
   ...
   </env:Header>
   <env:Body>
      <h:sayHelloToIdentity
            xmlns:h="http://soap.ncottin.net/hello">
         Hello Nathanael COTTIN!
      </h:sayHelloToIdentity>
   </env:Body>
</env:Envelope>
```

SOAP-RPC 1.2 binding: request example


```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
  envelope">
   <env:Header/>
   <env:Body>
      <h:sayHelloToIdentityMethod
            env:encodingStyle="http://www.w3.org/2003/05/
              soap/encoding/"
            xmlns:h="http://soap-rpc.ncottin.net/hello">
         <h:first>Nathanael</h:first>
         <h:last>COTTIN</h:last>
      </h:sayHelloToIdentityMethod>
   </env:Body>
</env:Envelope>
```


SOAP-RPC 1.2 binding: response example


```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-
  envelope">
   <env:Header>
      ...
   </env:Header>
   <env:Body>
      <my:sayHelloToIdentityResponse
            env:encodingStyle="http://schemas.xmlsoap.org/
              soap/encoding/"
            xmlns:my="http://soap-rpc.ncottin.net/hello">
         Hello Nathanael COTTIN!
      </my:sayHelloToIdentityResponse>
   </env:Body>
</env:Envelope>
```


WDDX 1.0 binding: overall format


```
<?xml version="1.0"?>
<wddxPacket version="1.0">
   <header><comment>WDDX example</comment><header>
   <data>
      <struct>
         <var name="param1">
            <string>Value</string>
         </var>
         <var name="param2">
            <array length="3">
               <number>-12.55</number>
               <boolean value="true"/>
               <null/>
            </array>
         </var>
      </struct>
   </data>
</wddxPacket>
```


WDDX 1.0 binding: request example


```
<?xml version="1.0"?>
<wddxPacket version="1.0">
   <header/>
   <data>
      <struct>
         <var name="first">
            <string>Nathanael</string>
         </var>
         <var name="last">
            <string>COTTIN</string>
         </var>
      </struct>
   </data>
</wddxPacket>
```

WDDX 1.0 binding: response example


```
<?xml version="1.0"?>
<wddxPacket version="1.0">
   <header/>
   <data>
      <string>Hello Nathanael COTTIN!</string>
   </data>
</wddxPacket>
```


JSON binding: request / response example

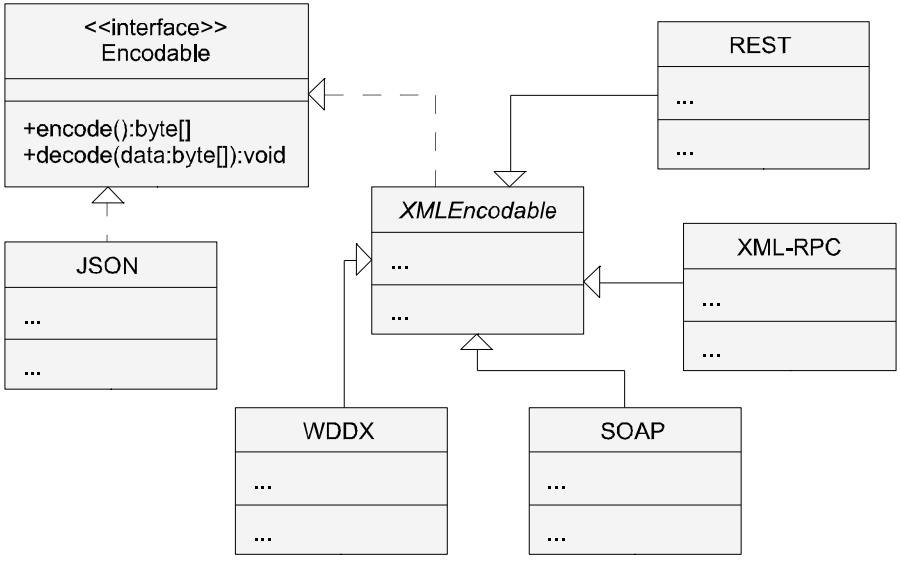
Request:

```
{
   "sayHelloToIdentity": {
      "first": "Nathanael",
      "last": "COTTIN"
   }
}
```

Response:

```
{
   "sayHelloToIdentityResponse": "Hello Nathanael COTTIN!"
}
```


Bindings summary 1/3



Bindings summary: bindings over HTTP 2/3

Request:

```
<meth> <url> HTTP/<version>
Host: www.ncottin.net
[Content-Length: <cLength>]
...
[Protocol-specific headers]
[
<Encoded request data>]
```

Response:

```
HTTP/<version> <status info>
Host: www.ncottin.net
Content-Length: <cLengh>
Content-Type: <cType>

<Encoded response data>
```

Bindings summary: bindings over SMTP 3/3


```
From: <senderAddress>
To: <recipientAddress>
Subject: <description>
Date: <currentDate>
Content-Type: <cType>
Content-Length: <cLength>
MIME-Version: 1.0
Message-Id: <ID>
[In-reply-to: <previousMessageId>
...
[Protocol-specific headers]

[<Encoded message data>]
```


Part 4

Web services description

- WSDL: purpose, description

WSDL overview

Model and XML format for describing web services

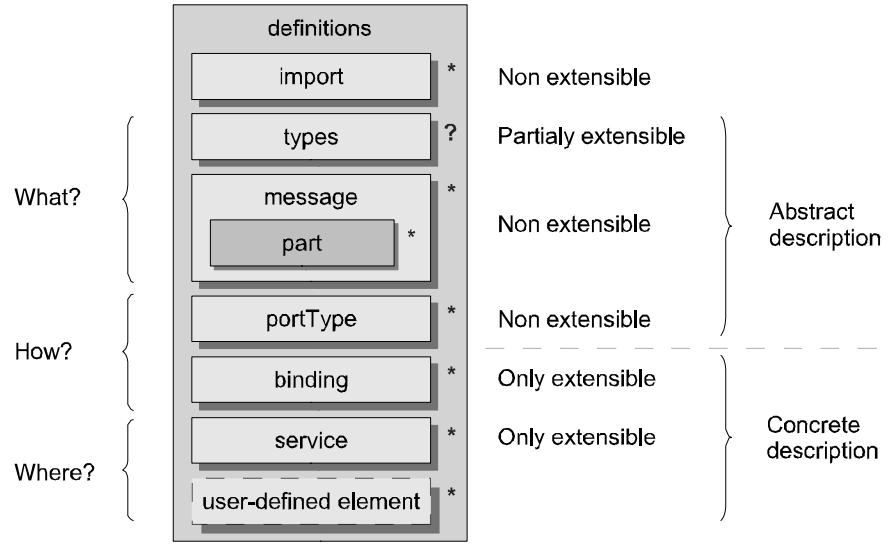
WSDL statically specifies:

- **What?**
- **How?**
- **Where?**

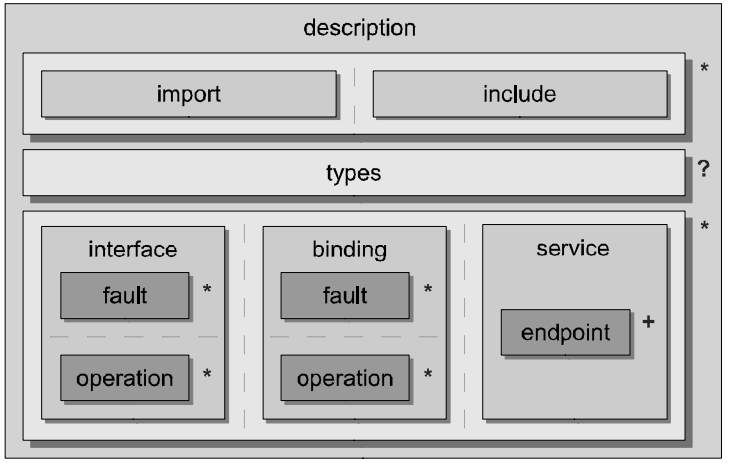
WSDL transmission primitives

- **One-way**: no response expected
- **Request / Response**: traditional exchange
- **Sollicit response**: callback with response
- **Notification**: callback without response

WSDL 1.1 overall format



WSDL 2.0 overall format



WSDL 2.0 extensibility element

Mandatory / optional extensions

Extension elements use "wsdl:required" attribute infoset

XSD:

```
<xs:complexType
      name="ExtensionElement"
      abstract="true">
   <xs:attribute
      ref="wsdl:required"
      use="optional"/>
</xs:complexType>
```


Part 5

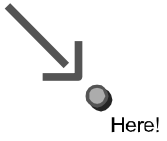
Web services registration and discovery

- UDDI
- BE4WS
- WSIL

About data discovery

Where are you?

...



"Data is worthless if it is lost within a mass of other data and cannot be distinguished or discovered"

OASIS UDDI Spec

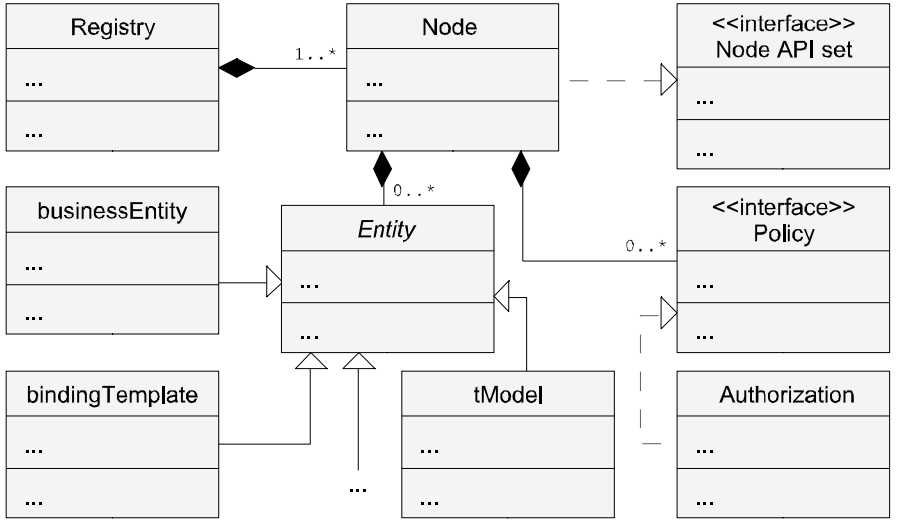
UDDI history

- **UDDI 1.0** **09/2000**
  - Originally announced by Ariba, Microsoft and IBM
  - UDDI initiative now includes more than 300 companies
- **First UDDI agents** **05/2001**
  - From Microsoft and IBM
- **UDDI 2.0** **06/2001**
  - Sponsored by OASIS
- **UDDI 3.0** **02/2005**

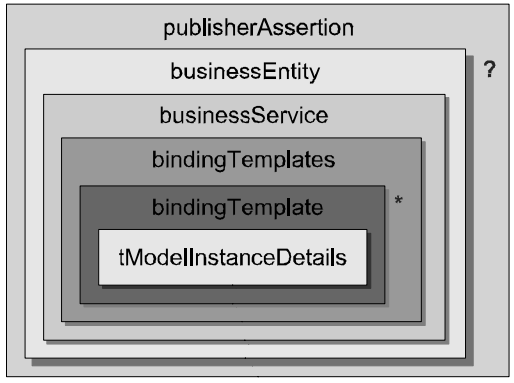
UDDI challenges

- **Availability**
- **Search results relevance**
- **Internationalization**
- **Security**
- **Synchronization**

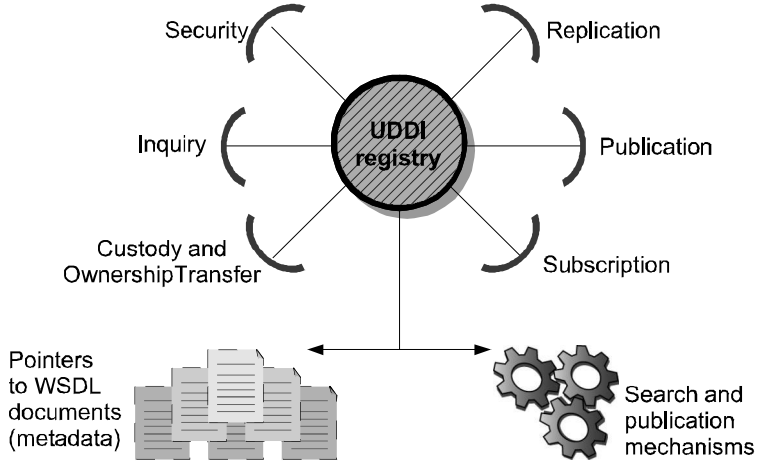
UDDI main components



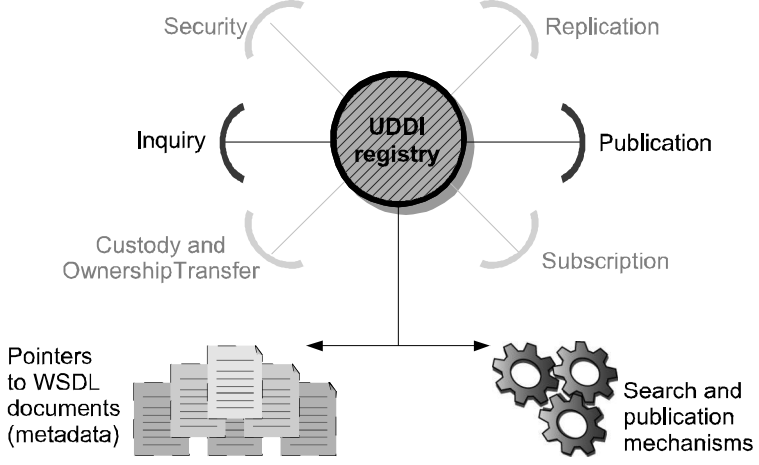
UDDI node overall format



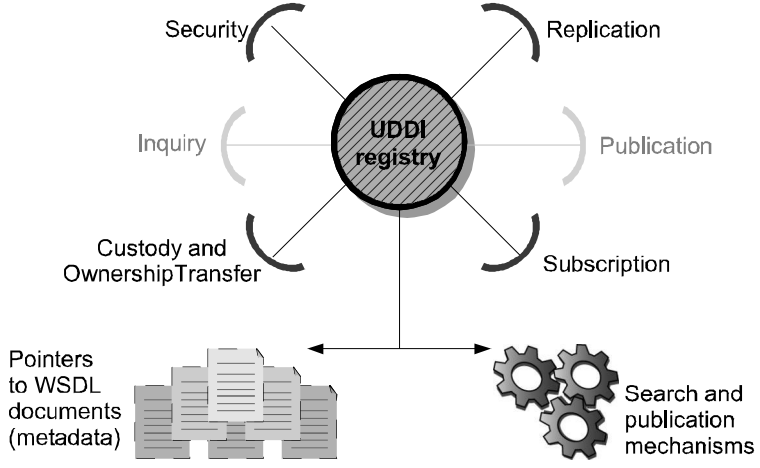
UDDI server-side APIs: overview



UDDI server-side APIs: main APIs



UDDI server-side APIs: complementary APIs



UDDI server-side API sets: inquiry

Request element:	Response element:
• find_binding	• bindingDetail
• find_business	• businessList
• find_relatedBusinesses	• relatedBusinessesList
• find_service	• serviceList
• find_tModel	• tModelList

UDDI server-side API sets: inquiry "find" example

find_business request:

```
<find_business xmlns="urn:uddi-org:api_v3" maxRows="10">
    <name xml:lang="en">Some business</name>
</find_business>
```

Corresponding response:

```
<businessList xmlns="urn:uddi-org:api_v3">
    <businessInfos>
        <businessInfo businessKey="892AC280-C16B-11D5-...">
            <name xml:lang="en">Some business</name>
            <description xml:lang="en">Business description</description>
            <serviceInfos>
                <serviceInfo serviceKey="..." businessKey="...">
                    <name xml:lang="en">Business service name</name>
                </serviceInfo>
            </serviceInfos>
        </businessInfo>
    </businessInfos>
</businessList>
```


UDDI server-side API sets: publication (excerpt)

Request element:	Response element:
• save_binding	• bindingDetail
• delete_binding	• <empty message> or fault
• add_publisherAssertions	• <empty message> or fault
• save_service	• serviceDetail
• delete_service	• <empty message> or fault

UDDI server-side API sets: publication "save" (creation) example

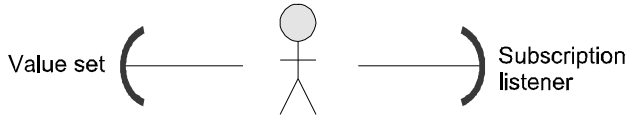
save_service request:

```
<save_service xmlns="urn:uddi-org:api_v3">
    <businessService
            serviceKey="40A5B22E-1A2B-5FAD-..."
            businessKey="892AC280-C16B-11D5-..."> <!-- If new service -->
        <name xml:lang="en">Some business</name>
        <description xml:lang="en">Business description</description>
    </businessService>
</save_service>
```

Corresponding response:

```
<serviceDetail xmlns="urn:uddi-org:api_v3">
    <businessService>
        <name xml:lang="en">Some business</name>
        <description xml:lang="en">Business description</description>
    </businessService>
</serviceDetail>
```

UDDI client-side APIs



UDDI client-side API sets: value set

Request element:

- `get_allValidValues`
- `validate_values`

Response element:

- `dispositionReport`
- `<empty message>` or `dispositionReport`

UDDI client-side API sets: subscription listener

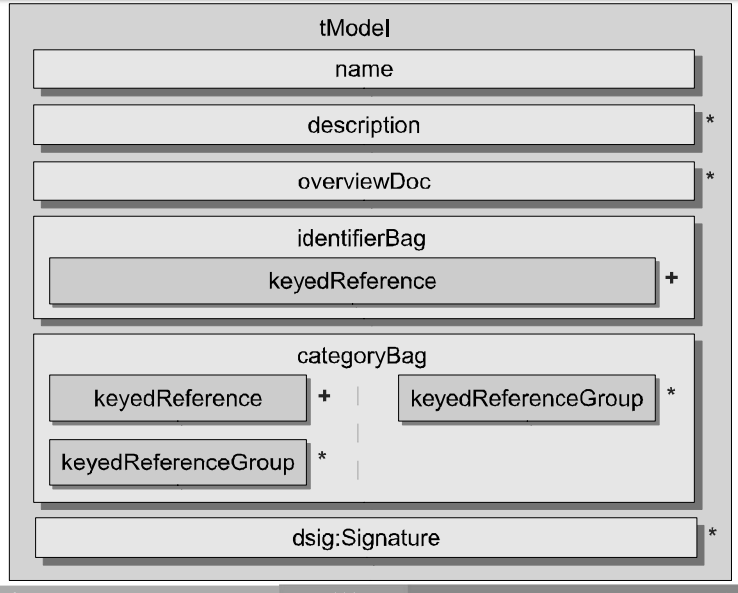
Request element:

- `notify_subscriptionListener`

Response element:

- `<empty message>`

UDDI "tModel" overall format



UDDI "tModel" example: inquiry results ordering


```
<tModel
  tModelKey="uddi:uddi.org:findqualifier:sortbynameasc">
   <name>uddi-org:sortByNameAsc</name>
   <description>UDDI sort qualifier...</description>
   <overviewDoc>
      <overviewURL useType="text">
         http://uddi.org/pubs/uddi_v3.htm#nameAsc
      </overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference
         keyName ="uddi-org:types:findQualifier"
         keyValue="findQualifier"
         tModelKey="uddi:uddi.org:categorization:types"/>
   </categoryBag>
</tModel>
```


UDDI "tModel" example: inquiry results ordering usage

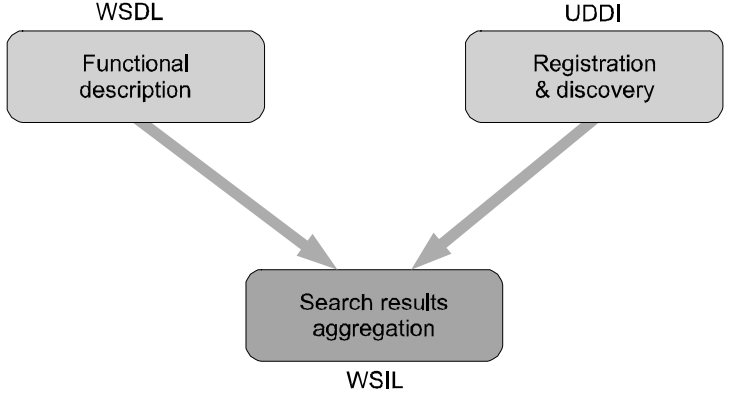

```
<find_business xmlns="urn:uddi-org:api_v3">
   <findQualifiers>
      <findQualifier>
         uddi:uddi.org:findqualifier:sortbynameasc
      </findQualifier>
   </findQualifiers>
   ...
   <name xml:lang="en">Some business</name>
</find_business>
```


Keyed references examples

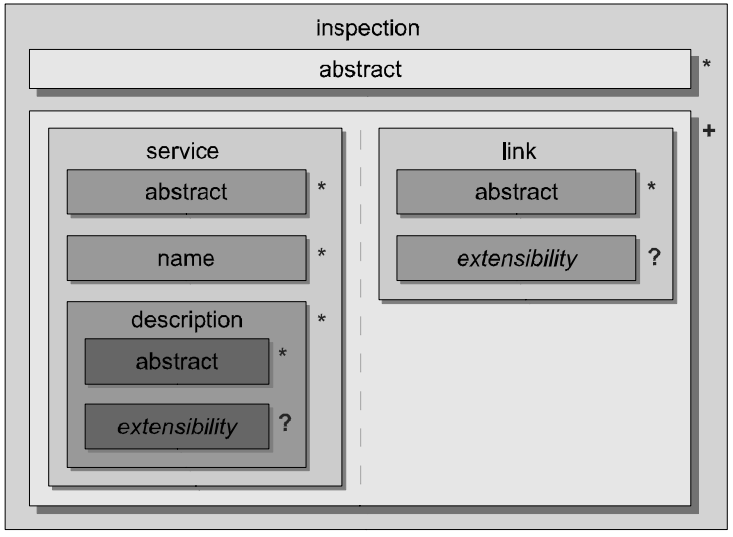

```
<keyReference
      tModelKey="uddi:ncottin.net:..."
      keyName="My key"
      keyValue="Any value"/>


<keyedReferenceGroup
      tModelKey="...">
   <keyedReference>...</keyedReference>
   ...
   <keyedReference>...</keyedReference>
</keyedReferenceGroup>
```


WSIL overview

- **IBM and Microsoft initiative**
- **XML grammar to tie WSDL and UDDI together**



WSIL overall format

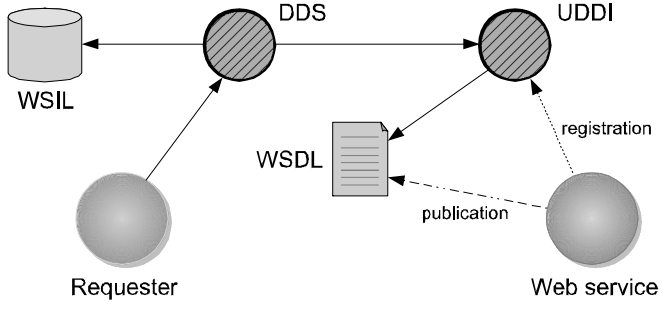


WSIL basic example

```
<?xml version="1.0"?>
<inspection xmlns="http://example.ncottin.net/inspection/">
   <abstract>Provider company name</abstract>
   <service>
      <name>Web service name</name>
      <description
          referencedNamespace="http://schemas.xmlsoap.org/wsdl/"
          location="http://example.ncottin.net/test.wsdl"/>
   </service>
</inspection>
```


Relationships between UDDI and WSIL to obtain WSDL

- **UDDI** registries provide pointers towards WSDL documents
- **WSIL** documents aggregate references to websites (including UDDIs) publishing WSDL documents
- **WSDL** documents describe web services interfaces



BE4WS overview

- **Portal for web services**
- **Extends WSIL capabilities**
- **Own language (USML) to build search queries**
- **Realized on BE4WS web services**
- **2 types of APIs:**
  - Regular API
  - Web services interface

BE4WS USML script example


```
<?xml version="1.0"?>
<!DOCTYPE Search SYSTEM "UDDISearch.dtd">
<Search>
   <ProcessId>12345</ProcessId>
   <Query>
      <Source>UDDI registry 1</Source>
      <SourceURL>http://uddi1_url</SourceURL>
      <BusinessName>name1</BusinessName>
      <FindBy>Business</FindBy>
   </Query>
   <Query>
      <Source>UDDI registry 2</Source>
      <SourceURL>http://uddi2_url</SourceURL>
      <BusinessName>name2</BusinessName>
      <FindBy>Business</FindBy>
   </Query>
   <AggOperator>OR</AggOperator>
</Search>
```


Part 6

Security considerations

- Need for secure web services
- Common and web services specific security issues
- Standards: WS-Security, REL, SAML, XACML, xmldsig
- Focus on security over SOAP binding

Need for secure web services

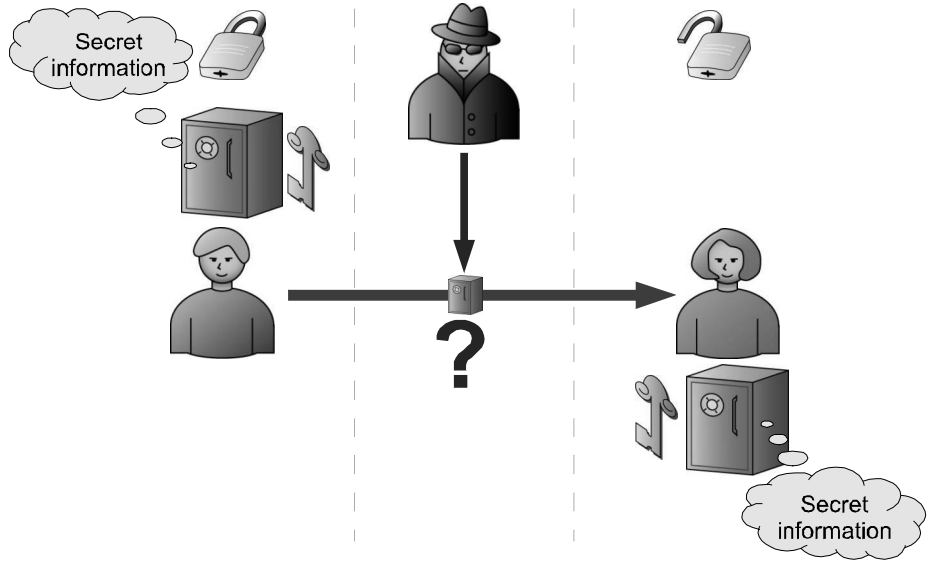
HTTPS protocol:

- **Confidentiality**
- **Integrity**
- **Authentication (destinatory and/or sender)**

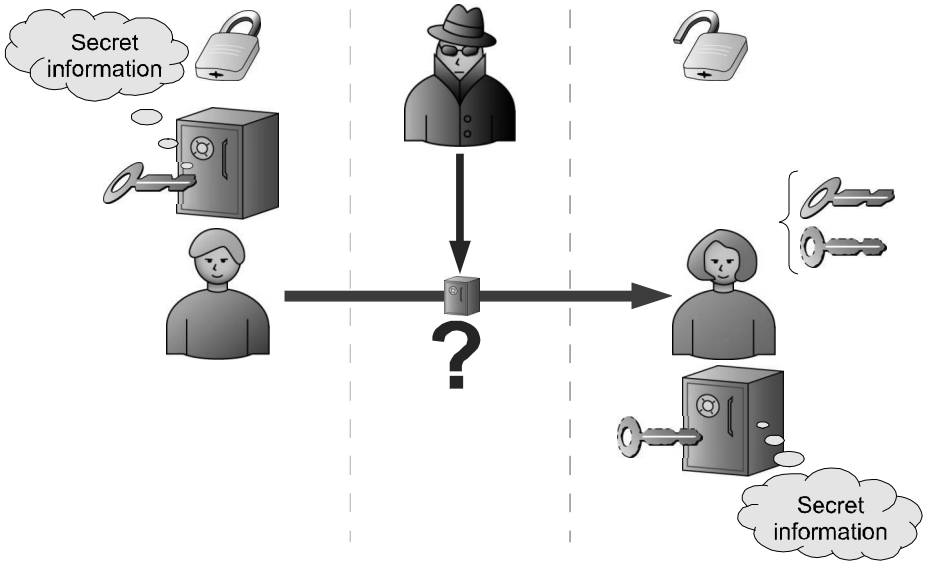
} **of the message**

Transport-level security not sufficient when intermediaries involved

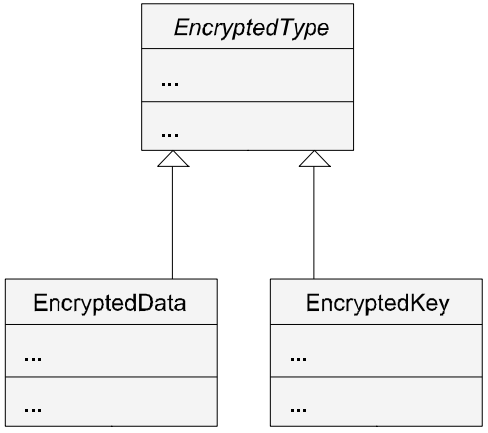
Symmetric key encryption principles



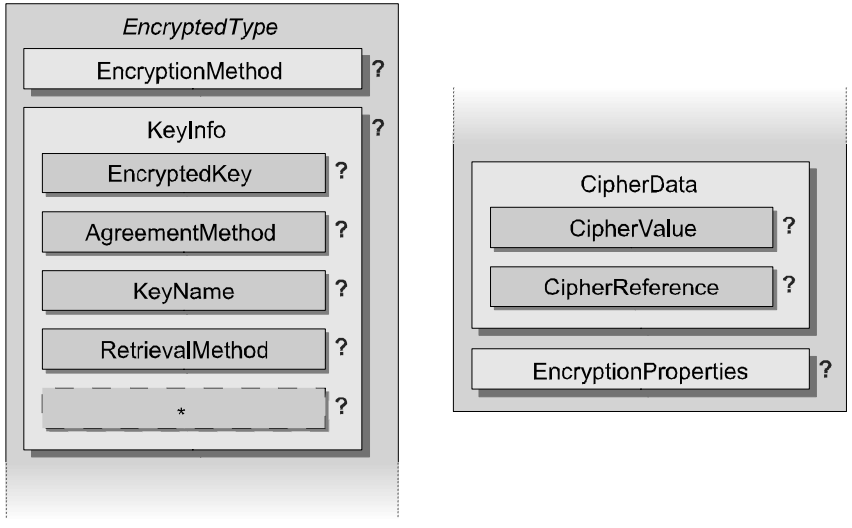
Asymmetric key encryption principles



XML encryption root types



XML encryption "EncryptedType" overall format



XML encryption targets

XML encryption applies to:

- **Simple types values**
- **Complex types inner types**
- **Documents**
- **Arbitrary data**

XML simple type content encryption example

Original (unencrypted) simple type:

```
<Number>1234 5678 9012</Number>
```

Encrypted simple type value:

```
<Number>
   <EncryptedData
     xmlns="http://www.w3.org/2001/04/xmlenc#"
     Type="http://www.w3.org/2001/04/xmlenc#Content">
      <CipherData>
         <CipherValue>45A6BC017F3B</CipherValue>
      </CipherData>
   </EncryptedData>
</Number>
```


XML complex type content encryption example

Original (unencrypted) complex type:

```
<CreditCard>
   <Number>...</Number>
   <Holder>...</Holder>
   <Issuer>...</Issuer>
</CreditCard>
```

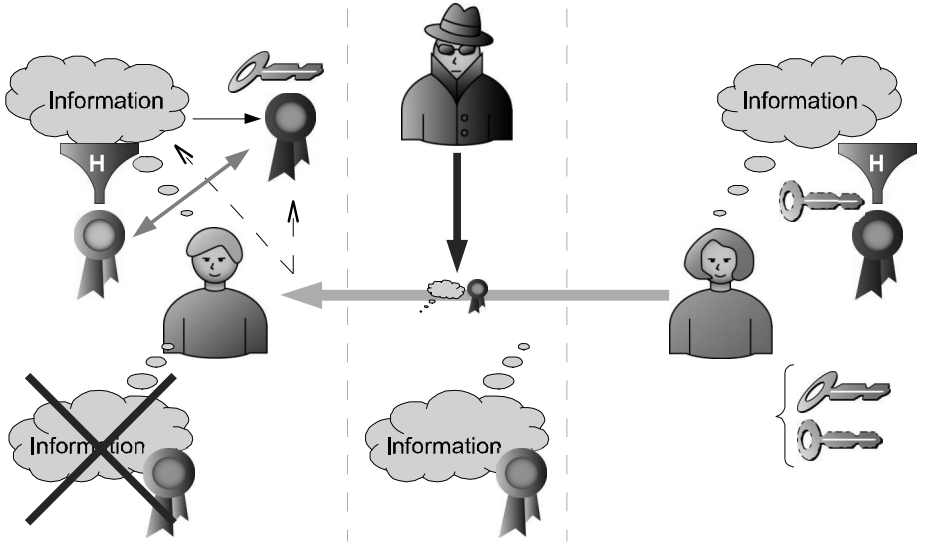
Encrypted complex type content:

```
<CreditCard>
   <EncryptedData
     xmlns="http://www.w3.org/2001/04/xmlenc#"
     Type="http://www.w3.org/2001/04/xmlenc#Content">
      <CipherData>
         <CipherValue>...</CipherValue>
      </CipherData>
   </EncryptedData>
</CreditCard>
```

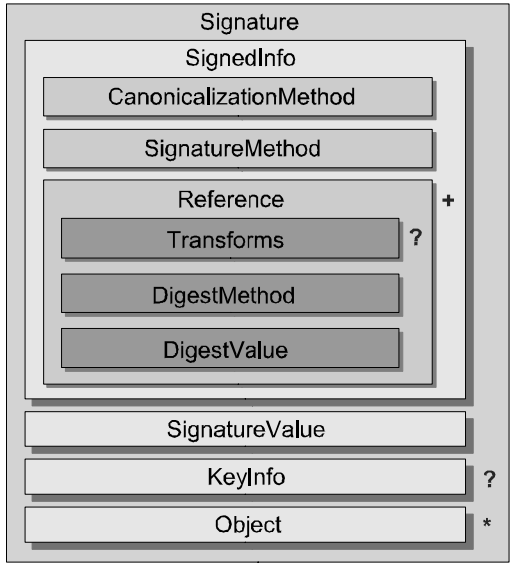

XML document or arbitrary data encryption example


```
<?xml version="1.0"?>

<EncryptedData
   xmlns="http://www.w3.org/2001/04/xmlenc#"
   MimeType="text/xml">
      <CipherData>
         <CipherValue>...</CipherValue>
      </CipherData>
</EncryptedData>
```

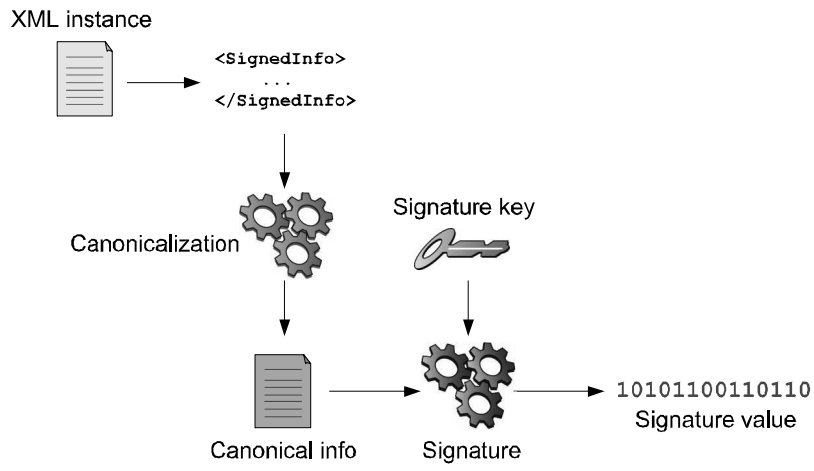

Digital signature principles



XML digital signature overall format



XML digital signature process



XML signature generation process

1. **Apply transforms to source data object**
2. **Calculate digest over resulting object**
3. **Create `Reference` element (may enclose optional `Id`)**
4. **Create `SignedInfo` (includes `Reference`)**
1. **Canonicalize `SignedInfo`**
1. **Create `SignatureValue` over canonicalized `SignedInfo`**
2. **Construct `Signature` element**

XML signature validation process

1. **Canonicalize `SignedInfo`**
1. **For each `Reference` element in `SignedInfo` do**
   a. **Obtain source data object (using `URI` attribute for example)**
   b. **Apply transforms to data object**
   c. **Digest transformed data object**
   d. **Compare generated digest against `DigestValue`**
2. **Obtain signature validation key using `KeyInfo`**
3. **Canonicalize `SignedInfo`**
4. **Validate `SignatureValue` over canonicalized `SignedInfo`**

WS-Security overview

- **Adds security information mainly using SOAP header:**

  `Security` element

- **Relies on existing standards:**
  - XML encryption
  - XML signature
  - Security tokens:
    - Kerberos
    - X.509 (PKI)
- **Specifies mechanisms for transferring:**
  - Simple user credentials
  - Encryption and signature tokens

"Security" element definition

- **Generic XML element:**
  - Sequence of any number of sub-elements
  - All namespaces authorized
  - XML Schema "lax" validation
- **Holds:**
  - Signatures
  - Keys references
  - Security tokens (credentials, binary tokens, **...**)

Tokens declarations: user credentials and raw data tokens


```
<UsernameToken>
   <Username>...</Username>
   <Passowrd>...</Password>
   <Nonce>...</Nonce>
   ...
</UsernameToken>

<BinarySecurityToken

   Id="..."
   ValueType="..."
   EncodingType="...#Base64Binary">
   ...

</BinarySecurityToken>
```


Tokens declarations: security tokens referencing

Using key identifier:

```
<wsse:SecurityTokenReference Id="..." TokenType="..."
 Usage="...">
   <wsse:KeyIdentifier Id="..."
    EncodingType="...#Base64Binary" ValueType="...">
    ...
   </wsse:KeyIdentifier>
</wsse:SecurityToken>
```

Direct reference:

```
<wsse:SecurityTokenReference Id="..." TokenType="..."
 Usage="...">
   <ds:Reference URI="..." ValueType="..."/>
</SecurityToken>
```

X.509 security token: token types 1/4

- **Single X.509 v3 certificate**
- **X.509 certificate path**
- **Set of X.509 certificates and CRLs**

X.509 security token: Subject Key Identifier reference 2/4


```
<wsse:Security xmlns:wsse="...">
   <ds:Signature xmlns:ds=".../xmldsig#">
      <ds:SignedInfo>
         ...
         <ds:Reference URI="#keyId">...</ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo Id="keyId">
         <wsse:SecurityTokenReference>
            <wsse:KeyIdentifier
              EncodingType="...#Base64Binary"
              ValueType="...#X509SubjectKeyIdentifier">
                ...
            </wsse:KeyIdentifier>
         </wsse:SecurityTokenReference>
      </ds:KeyInfo>
   </ds:Signature>
</wsse:Security>
```


X.509 security token: Binary Security Token reference 3/4


```
<wsse:Security xmlns:wsse="..." xmlns:wsu="...">
   <wsse:BinarySecurityToken wsu:Id="certificateId"
     ValueType="...#X509v3" EncodingType="...#Base64Binary">
      ...
   </wsse:BinarySecurityToken>
   <ds:Signature xmlns:ds=".../xmldsig#">
      <ds:SignedInfo>
         ...
         <ds:Reference
           URI="#certificateId">...</ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo>
         <wsse:SecurityTokenReference>
            <wsse:Reference URI="#certificateId"/>
         </wsse:SecurityTokenReference>
      </ds:KeyInfo>
   </ds:Signature>
</wsse:Security>
```


X.509 security token: Issuer and Serial Number reference 4/4


```
<wsse:Security xmlns:wsse="...">
   <ds:Signature xmlns:ds=".../xmldsig#">
      <ds:SignedInfo>
         ...
         <ds:Reference URI="#keyId">...</ds:Reference>
      </ds:SignedInfo>
      <ds:SignatureValue>...</ds:SignatureValue>
      <ds:KeyInfo Id="keyId">
         <wsse:SecurityTokenReference>
            <ds:X509Data>
               <ds:X509IssuerSerial>
                  <ds:X509IssuerName>...</ds:IssuerName>
                  <ds:X509SerialNumber>...</ds:509SerialNumber>
               </ds:X509IssuerSerial>
            </ds:X509Data>
         </wsse:SecurityTokenReference>
      </ds:KeyInfo>
   </ds:Signature>
</wsse:Security>
```

Kerberos security token: token types

- **AP requests**
- **GSS AP requests**

SOAP message security concepts

Use of SOAP `Header` element to include `Security` element

SOAP message-level security:

- **Confidentiality**
- **Integrity**
- **Authentication**

SOAP-based encryption example


```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="..." xmlns:xenc="...">
   <soap:Header>
      <wsse:Security xmlns:wsse="...">
         <xenc:ReferenceList>
            <xenc:DataReference URI="#ref"/>
         </xenc:ReferenceList>
      </wsse:Security>
   </soap:Header>
   <soap:Body>
      <xenc:EncryptedData Id="ref" Type="...#Content">
         <xenc:EncryptionMethod Algorithm="..."/>
         <xenc:CipherData>
            <xenc:CipherValue>...</xenc:CipherValue>
         </xenc:CipherData>
      </xenc:EncryptedData>
   </soap:Body>
</soap:Envelope>
```

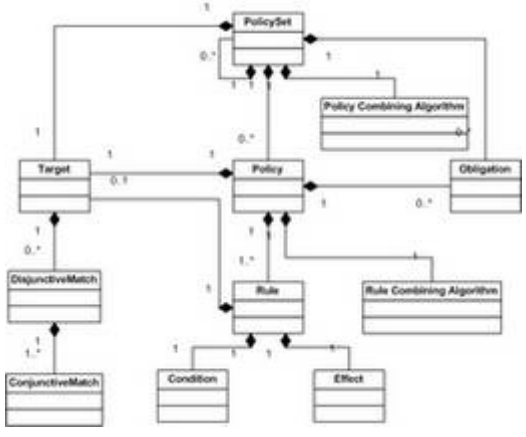

SOAP-based digital signature example


```
<soap:Envelope xmlns:soap="...">
   <soap:Header>
      <wsse:Security xmlns:wsse="...">
         <ds:Signature Id="signId"
           xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
               <ds:CanonicalizationMethod Algorithm="..."/>
               <ds:SignatureMethod
                  Algorithm=".../xmldsig#dsa-sha1"/>
               <ds:Reference URI="#soapBodyId">
                  ...
               </ds:Reference>
            </ds:SignedInfo>
            <ds:SignatureValue>...</ds:SignatureValue>
         </ds:Signature>
      </wsse:Security>
   </soap:Header>
   <soap:Body Id="soapBodyId">
      ...
   </soap:Body>
</soap:Envelope>
```

SAML 2.0 overall format

XACML objectives

XACML 3.0 overall format



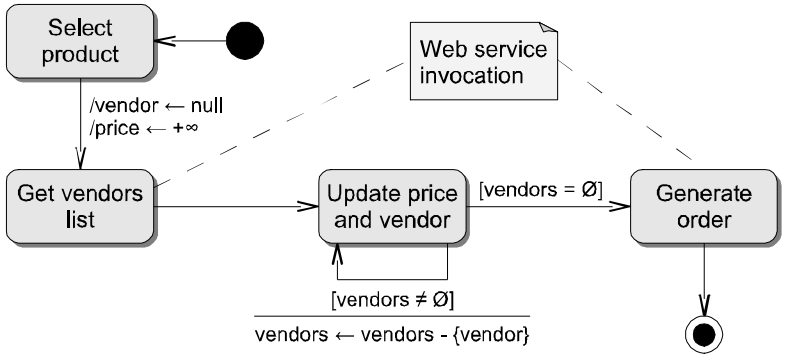
Part 7

Even more on web services

- Orchestration (BPEL) and choreography (WSCI)
- WS-*

Need for orchestration

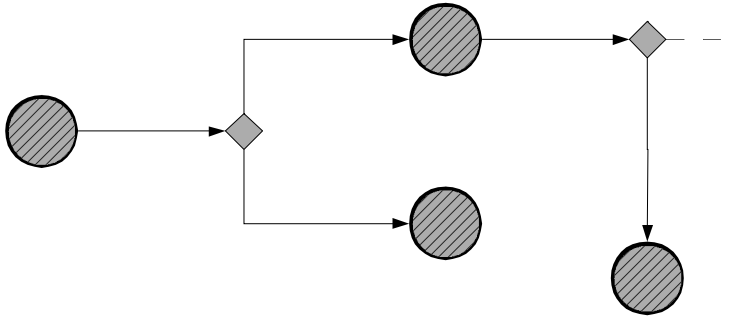
- **Problem: select the lowest price product from different vendors and generate a purchase order**
- **Algorithm (simplified):**



Web services orchestration with BPEL

- **XLANG (Microsoft)**
- **WSFL (IBM)**

} **BPEL**

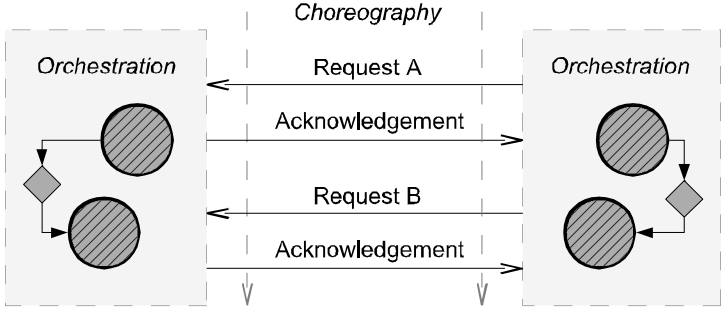


Orchestration and choreography

The difference is currently an open W3C issue

Orchestration refers to an executable process

Choreography = MEPs composition + application semantics



WS-Inspection

- **To replace WSIL and other initiatives (DISCO, ADS, ...)**
- **XLANG + WSFL --> WSCL**

Conclusion

Web services: which future?

- Web services infrastructure normalization
- Web services pros and cons
- XML versus other encodings

Web services infrastructures normalizations

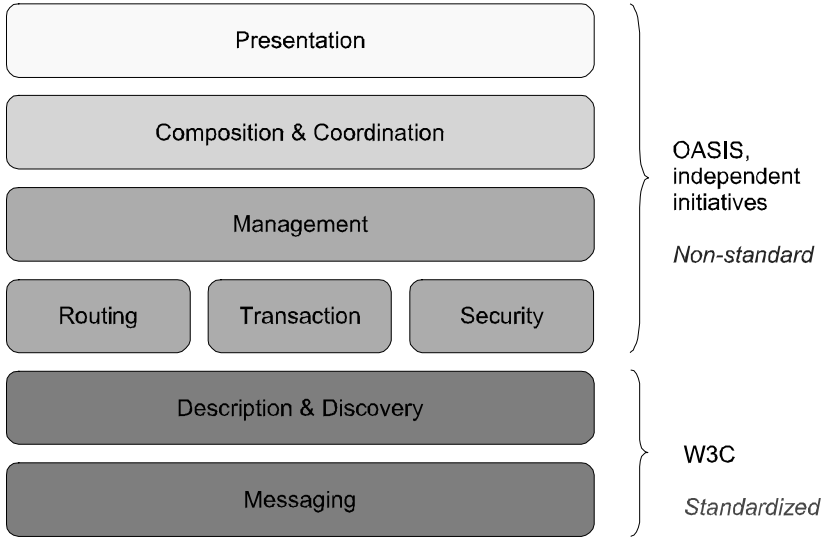
Horizontal (transversal) services:

- **Transport layer**: HTTP / SOAP
- **Semantics layer**: business data normalization, ebXML, UBL, RosettaNet
- **Process management layer**: BPML, XLANG, WSFL, BPEL

Vertical (business-oriented) common facilities:

- **Naming**: UDDI, WSIL
- **Transaction**: XAML, BTP
- **Security**: SAML, XACML

Web services general stack



GXA components

Web services: yet another marketing initiative?

Pros:

- **Well-known, widespread and improved technologies**
- **Robust implementations accessible for free**
- **Standardized core technologies (promoted by W3C and IETF)**
- **OASIS high-level standards**

Cons:

- **Security issues:**
  - HTML and XML common security issues
  - XML extensions to support security
  - Common security standards primarily defined using ASN.1 notation
- **Performances**: real-time systems, **...**
- **WS-* galaxy**
- **"Work-in-progress"**

Comparisons with other technologies

- **HTTP versus BEEP**
- **XML versus ASN.1 and its standard encodings BER, CER, DER, PER, XER**