Web services

Service-Oriented Architecture

Jeremy Gibbons

Contents

1 Web services

- implementing SOA using standard technologies:
  XML, HTTP; WSDL, UDDI, SOAP; WS-$*$
- term coined by Microsoft in 2000
- exploiting ubiquitous WWW infrastructure for distributed computing
- just one way to do SOA
- apologia: covered upfront here for logistical reasons
  (Josuttis relegates web services to Chapter 16)

1.1 World-wide web

- navigating document collections
- multimedia documents
- hypertext cross-references
- hypertext markup language (HTML)
- hypertext transfer protocol (HTTP)
- Tim Berners-Lee at CERN, 1989–1992

1.2 The evolving web

generation	access	technology	example
first	manual	browser	arbitrary HTML
second	programmatic	screen-scraper	systematic HTML
third	standardized	web service	formally described service
fourth	semantic	semantic WS	semantically described service

The *deep web* dominates the *surface web*.

1.3 Some definitions

A *web service* is an interface that describes a collection of operations that are network accessible through standardized XML messaging. A web service is described using a standard, formal XML notion, called its *service description*. It covers all the details necessary to interact with the service, including message formats (that detail the operations), transport protocols and location. The interface *hides the implementation* details of the service, allowing it to be used independently of the hardware or software platform on which it is implemented and also independently of the programming language in which it is written. This allows and encourages WS-based applications to be *loosely coupled, component-oriented, cross-technology* implementations.

(IBM Web Services Conceptual Architecture, 2001)

The basic idea behind web services is to adapt the *loosely coupled web programming model* for use in applications that are not browser-based. The goal is to provide a platform for building distributed applications using software running on different operating systems and devices, written using different programming languages and tools from multiple vendors, all potentially developed and deployed independently.

(Understanding XML Web Services: The Web Services Idea. Tim Ewald, Microsoft, 2002)

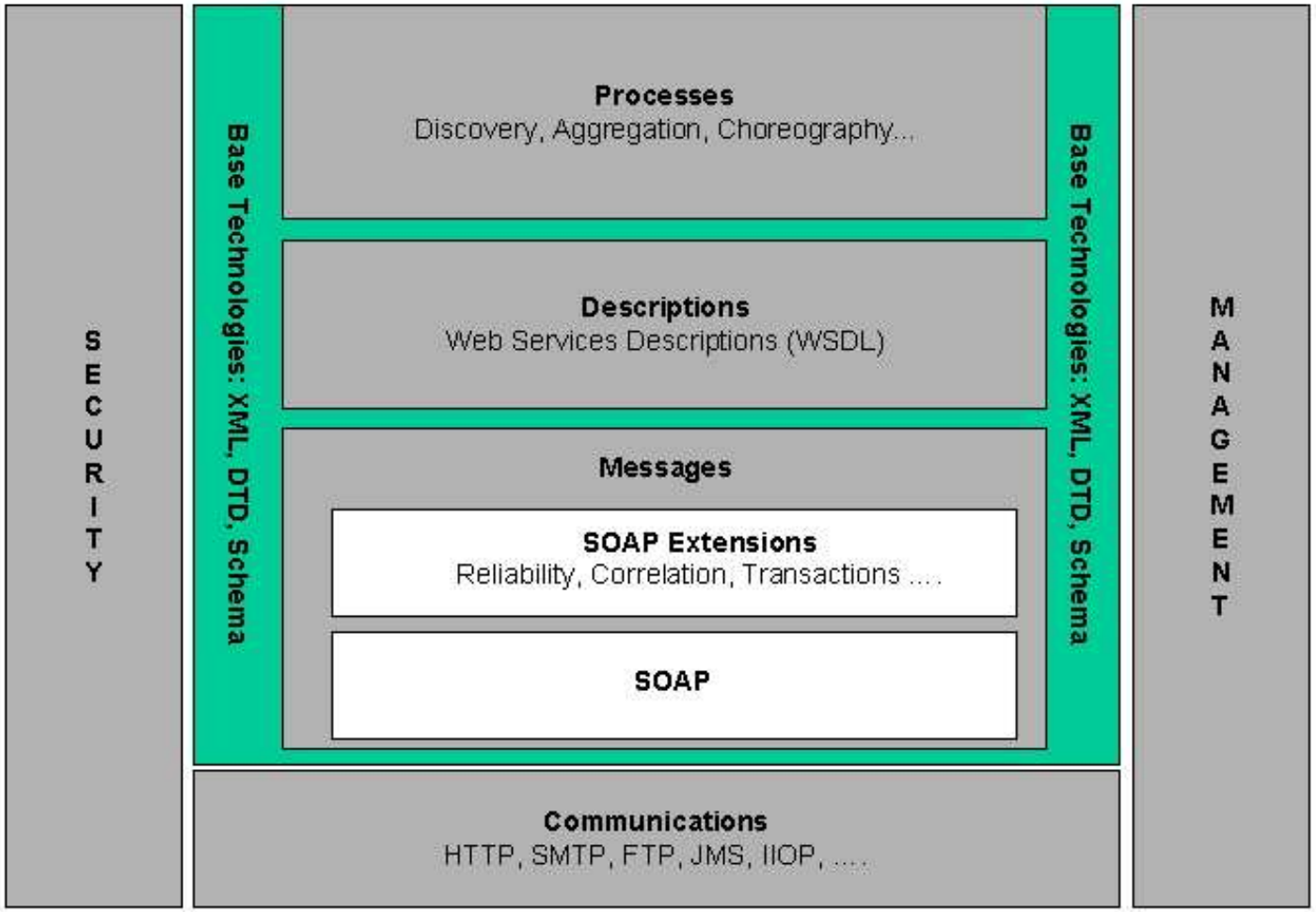
A Web service is any executable code (program) that can be *discovered, described, and invoked* using XML-based messages via Internet-based protocols. [. . .] Today, the leading XML-based standards are UDDI for discovery, WSDL for description, as well as SOAP and ebXML for XML message handling and service invocation.

(Sun ONE FAQ)

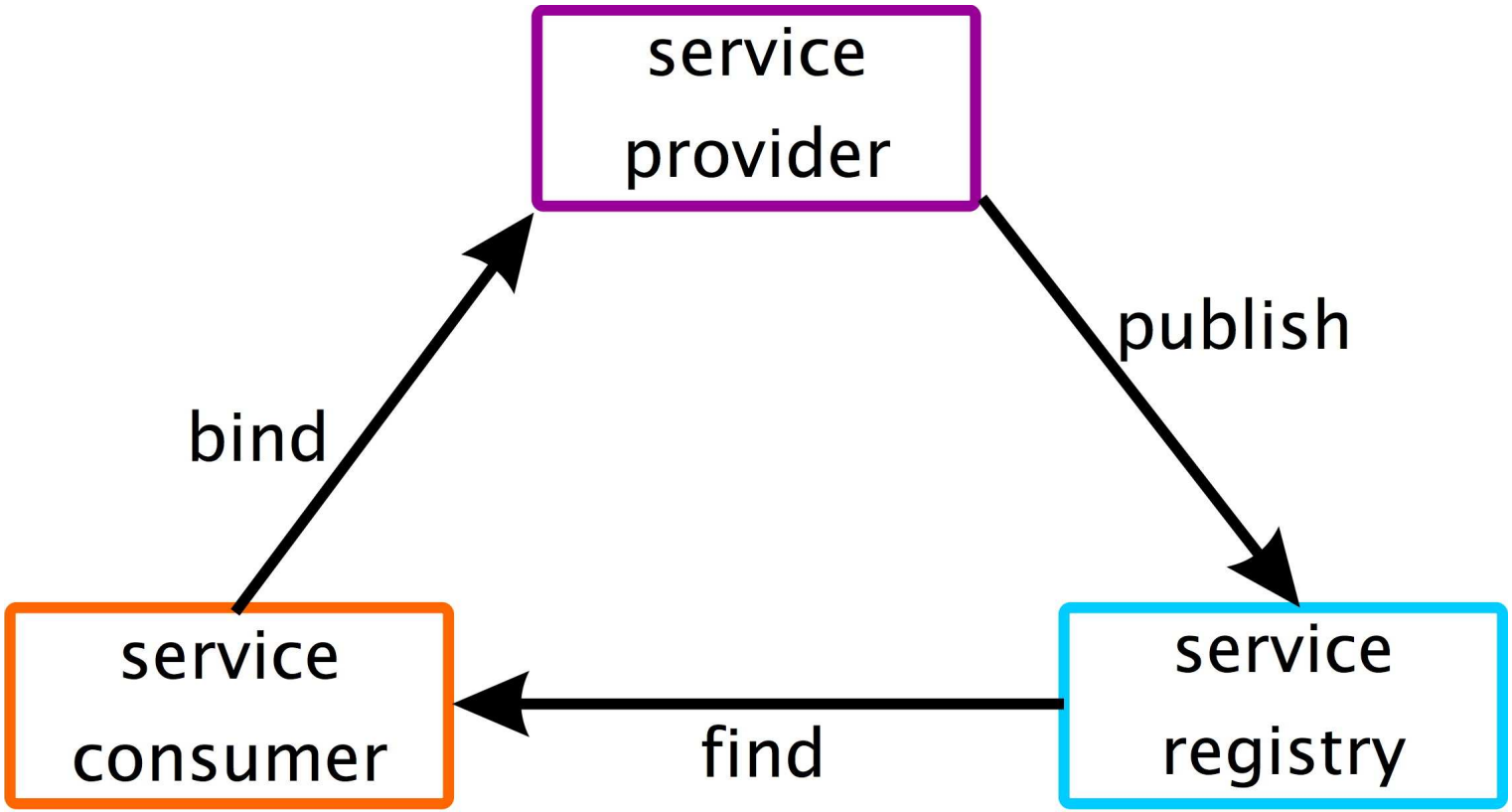
A web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards.

(http://www.w3.org/TR/ws-arch/#whatis)

1.4 Web services architecture (W3C)



1.5 Publish–find–bind



1.6 Alphabet soup

SOAP	Simple Object Access Protocol
WSDL	Web Services Description Language
UDDI	Universal Description, Discovery, and Integration
HTTP	Hypertext Transfer Protocol
SSL/TLS	Secure Sockets Layer/Transport Layer Security
HTTPS	HTTP over SSL
WS-I	Web Services Interoperability
WSFL	Web Services Flow Language
WSIL	Web Services Inspection Language
WSCL	Web Services Co-ordination Language
BPEL4WS	Business Process Execution Language for Web Services

XML

XML	eXtensible Markup Language
DTD	XML Datatype definition language
XML-Schema	(specifying validity of XML documents)
XSLT	eXtensible Stylesheet Language for Transformations
XPath	XML Path Language
XML namespaces	(scoping of identifiers)

Grid

OGSI	Open Grid Services Infrastructure
OGSA	Open Grid Services Architecture
DAML	DARPA Agent Markup Language
XACML	eXtensible Access Control Markup Language
SAML	Security Assertion Markup Language

Java

JSF	JavaServer Faces
JAXB	Java Architecture for XML Binding
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries
JAX-RPC	Java API for XML-based RPC
SAAJ	SOAP with Attachments API for Java
JSTL	JavaServer Pages Standard Tag Library
WSDP	Sun Java Web Services Developer Pack
WSTK, ETTK	IBM Web Services Toolkit
AXIS	Apache SOAP implementation
WSIF	Web Services Invocation Framework

. . . and other languages

gSOAP	C++ Toolkit for Web Services
.NET	Microsoft; bindings for many languages
SOAPpy	Python Web Services
SOAP::Lite	Perl library for Web Services
HAIFA	Haskell library for Web Services
SOAP4R	Ruby implementation of SOAP 1.1

Organizations

OASIS	Organization for the Advancement of Structured Information Standards
W3C	World-Wide-Web Consortium
JSR	Java Standards
IETF	Internet Engineering Task Force
GGF	Global Grid Forum
WS-I	Web Services Interoperability Organization

WS-*

WS-Security WS-Policy WS-SecurityPolicy WS-PolicyAssertions
WS-PolicyAttachment WS-Trust WS-Privacy WS-Routing WS-Referral
WS-Coordination WS-Transaction WS-SecureConversation WS-Federation
WS-Authorization WS-Inspection WS-Attachments WS-ReliableMessaging
WS-Addressing

WS-Notification WS-ResourceProperties WS-ResourceLifetime
WS-RenewableReferences WS-ServiceGroup WS-BaseFaults

WS-CAF (composite applications); WS-CTX (context); WS-CF (coordination);
WS-TXM (transaction management)

1.7 A challenge

- Which ones exist?
- Which ones are *de facto* standards?
- Which ones are standardized?
- Which ones are being superseded?
- Which ones layer on which other ones?
- Which ones matter?

2 XML, in a nutshell

- IBM GML (1969) → SGML (1986) → HTML (1991), XML (1996)
- nested *tags*: tree structure, regular syntax, simple to parse
- tags may have *attributes* and surround *text*
- *well-formedness* vs *validity*
- constraints, typing: DTD, XML Schema, RELAX NG
- *namespaces* for qualified names, disambiguation
- Extensible Stylesheet Language Transformations (XSLT)
- XPath for *locations* within XML documents
- XQuery for *extracting fragments* of XML documents
- programmatic traversal: *event-based* SAX, *tree-based* DOM

3 HTTP

- two-way transmission of requests and responses
- layered over TCP
- essentially stateless (but. . .)
- standard extensions for security

3.1 Network layers

OSI model	*internet stack*
application presentation	application
session transport	TCP, UDP
network	IP
datalink physical	ethernet (802.3), wifi (802.11)

3.2 Methods

- GET *uri*

  read a document; should be “safe”

- PUT *uri*, *data*

  create or modify a resource; should be idempotent

- POST *uri*, *data*

  create a subordinate resource

- DELETE *uri*

  delete a resource; should be idempotent

(also HEAD, TRACE, OPTIONS, CONNECT)

3.3 Identifying resources

- *uniform resource identifier* (URI)
- *uniform resource locator* (URL)
- *uniform resource name* (URN)

For a large class of schemes, the syntax is

⟨*scheme*⟩`://`⟨*authority*⟩⟨*path*⟩`?`⟨*query*⟩

The *classical view* is that URIs are partitioned into URLs (which describe a primary access mechanism, eg `http:`) and URNs (which do not, eg `isbn:`, and need a separate resolver).

The *contemporary view* is that URIs may define subspaces; `http:` is a URI scheme, and `urn:isbn:` is a URN namespace. 'URL' is somewhat deprecated.

3.4 HTTP v1.1 request

⟨*method*⟩ ⟨*path*⟩ `HTTP/1.1`
⟨*headers*⟩
blank line
optional ⟨*body*⟩

- Accept, Accept-Encoding, Accept-Language, . . .
- Authorization (actually, it's authentication, obfuscated)
- Cookie data returned from previous response
- Host domain name, mandatory in v1.1
- If-Modified-Since
- Referer [sic]: URI of resource providing request URI
- User-Agent application/version
- Connection for keep-alive

3.5 Response

`HTTP/1.1` ⟨*code*⟩ ⟨*reason*⟩
⟨*headers*⟩
blank line
optional ⟨*body*⟩

- Content-Length, Content-Language, Content-Type...
- Date, mandatory in v1.1
- Last-Modified
- Set-Cookie
- Location for redirects (3xx)
- Retry-After for temporary unavailability (503)
- Keep-Alive

3.6 Status codes

100	Continue
200	OK
201	Created
202	Accepted
204	No Content
301	Moved Permanently
303	See Other
304	Not Modified
307	Temporary Redirect
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
404	Not Found
410	Gone
500	Internal Error
501	Not Implemented
503	Service Unavailable

3.7 Example: request

```
GET /index_nlp.html HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg,
  application/vnd.ms-powerpoint, application/vnd.ms-excel,
  application/msword, application/x-shockwave-flash, */*
Accept-Language: en-gb
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0...)
Host: portal.free-online.net
Connection: Keep-Alive
Cookie: Example_Session=40eabfeb7504452c3306fc2e46ceef01
```

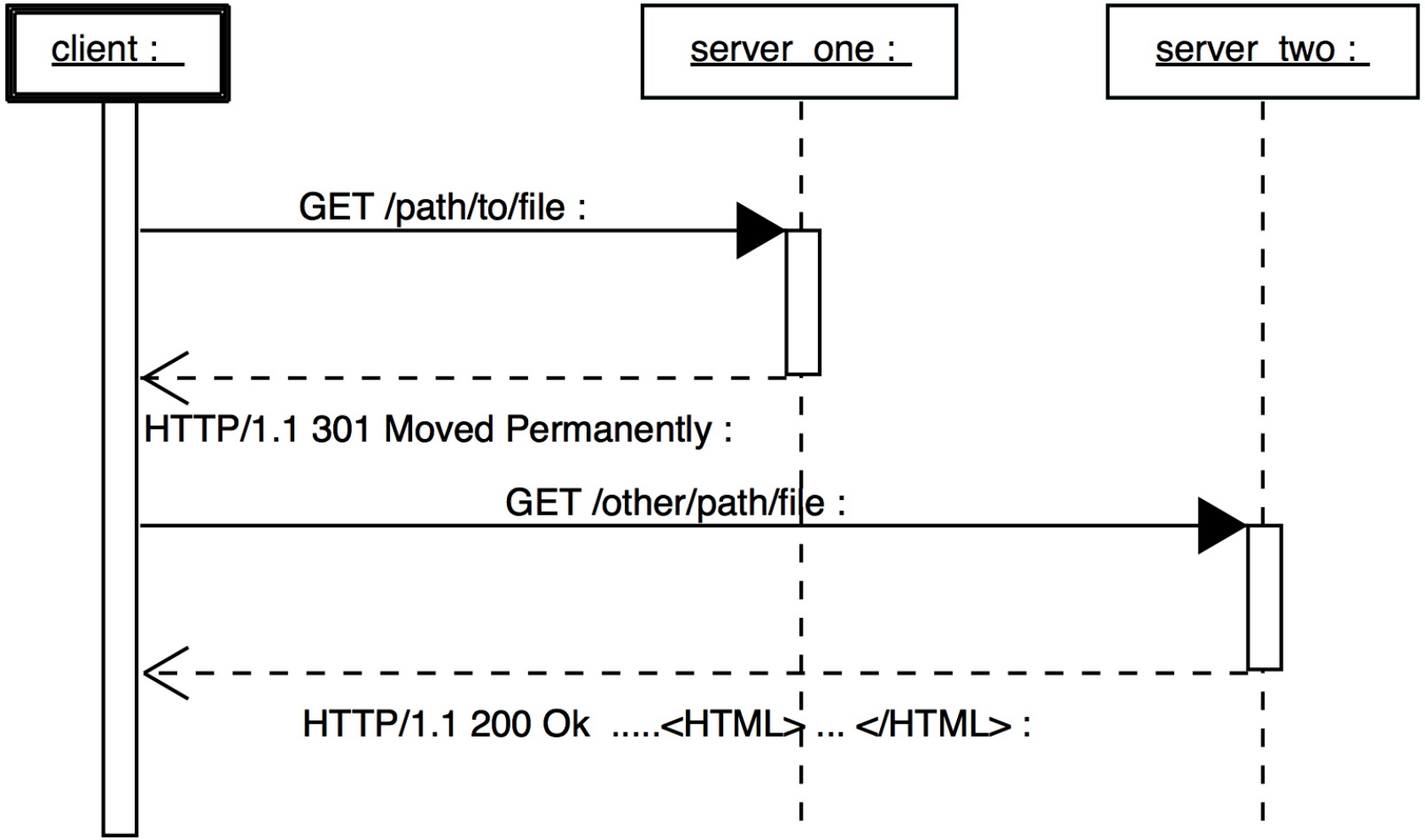
3.8 Example: response headers

```
HTTP/1.1 200 OK
Date: Thu, 03 Jul 2003 16:33:04 GMT
Server: Apache/1.3.26
X-Powered-By: PHP/4.0.6
X-Accelerated-By: PHPA/1.3.3r1
Keep-Alive: timeout=5, max=28
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
```

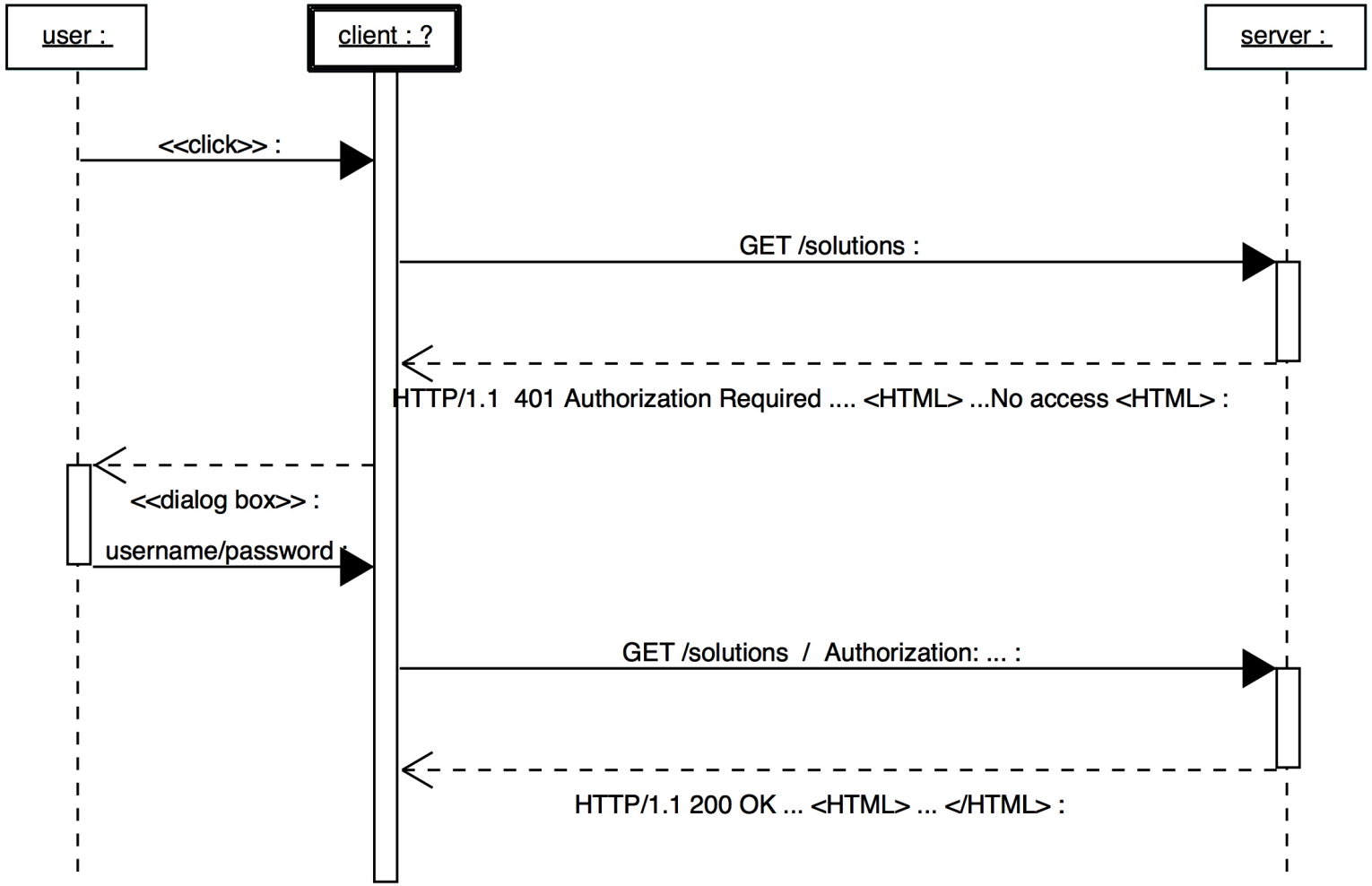
3.9 Example: response body

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>[ Free-Online ] - The easy to use internet service</TITLE>
<META content="text/html; charset=iso-8859-1">
...
</HEAD>
<body leftmargin="0" topmargin="0">
<!-- BEGIN HEADER -->
<table width="100\%" border="0" cellspacing="0" cellpadding="0">
...
</table>
</body>
</HTML>
```

3.10 Redirection



3.11 Authentication



4 Simple Object Access Protocol (SOAP)

- Envelope, consisting of Header and Body
- all this in XML
- serialization `http://schemas.xmlsoap.org/soap/encoding/`
- now just a name, not an acronym: not simple, and not about object access! (see *The 'S' Stands for 'Simple'*)

4.1 SOAP envelope structure

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encoding-style="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Header>
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.2 SOAP headers

- optional
- essentially free-form XML
- major extensibility capability
- a few attributes are pre-defined:
  - mustUnderstand attribute set to 1 for a header element that must be processed; recipient must abort if it cannot
  - SOAP-ENV:actor indicates URI for server (intermediary) which should process a given header; typically for overhead information: authentication, authorization, transaction management, trace, audit, routing
- logically the body could just be a header with `mustUnderstand="1"` (but it is not)

4.3 Example SOAP message

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope
  xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:enc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:ns0="http://hello.org/wsdl">
  <env:Body>
    <ns0:sayHelloBackResponse
      env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <result xsi:type="xsd:string">
        Hi JAXRPC Sample From Server for wsdl client
      </result>
    </ns0:sayHelloBackResponse>
  </env:Body>
</env:Envelope>
```

4.4 SOAP body

Arbitrary XML — but there is a standard format for fault reporting:

```
<SOAP-ENV:Fault>
  <faultcode> ... </faultcode>
  <faultstring> ... </faultstring>
  <faultactor> ... </faultactor>
  <detail> ... </detail>
</SOAP-ENV:Fault>
```

4.5 SOAP validation

- against schema
- against WSDL description
- through use of toolkits: developer never writes SOAP directly
- great scope for non-interoperability!

4.6 Processing model

Upon receiving a SOAP message, a SOAP application must:

1. determine whether it understands the version of SOAP detailed in the namespace attribute of the SOAP envelope — if not, must discard message with `VersionMismatch` error;
2. identify all parts of the message intended for the application (intermediary or final recipient);
3. verify mandatory parts of message, including `mustUnderstand` — else `mustUnderstand` error, or application error;
4. process all mandatory parts, plus optional parts it understands;
5. if not the final recipient, remove the headers it has processed before passing the message forward.

4.7 SOAP body: data models

- standardized encodings for complex datatypes
- SOAP envelope namespace defines `encodingStyle` attribute
- well-defined mapping between abstract data models (ADMs) and XML
- XMLSchema types, simple & compound types, structs, arrays
- careful treatment of references; inclusion of XML is easy; care with namespaces; more care with `id` attributes; binary data within `<SOAP-ENC:base64>`
- *serializers* and *deserializers* convert application data ↔ XML
- construction of XML can be left to the message creation stage, or can take place at a higher level in the application (compare RPC and document-centric approaches)
- encodings include: SOAP encoding; Literal XML; XMI

4.8 SOAP shortcomings

- simplistic assumption that data to be exchanged is method input/output parameters
- not all data is suitable for serialization into XML
- MIME attachments are added to SOAP, complexity results
- need for higher-level standards for security, reliable messaging, coordination, transaction, context, etc

5 Web Services Description Language (WSDL)

- key interface definition language for web services
- XML document(s), hence verbose
- essence of interoperability
- most toolkits generate WSDL from service code
- most toolkits generate client code from WSDL
- largely, then, a machine format!

5.1 WSDL components

types potentially heavily structured XML data

message (abstractly) one for each interaction; 0 or more parts in each

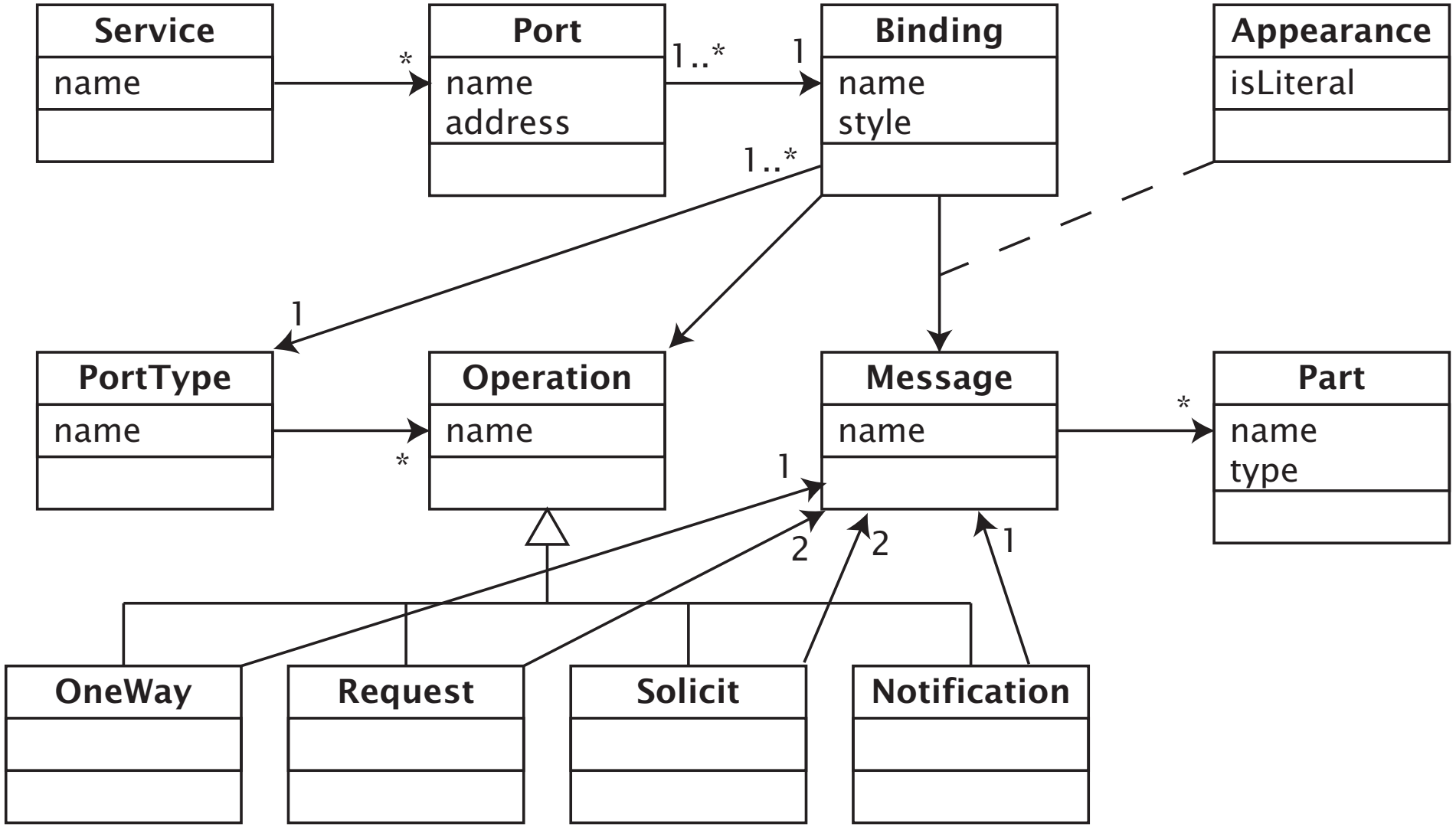
operation abstract operation/method, defined by its messages

portType set of abstract operations

binding joins operations to transport protocols

port endpoint; address for binding

service collection of related ports


Service
name
Port
name
address
Binding
name
style
Appearance
isLiteral
PortType
name
Operation
name
Message
name
Part
name
type
OneWay
Request
Solicit
Notification
*
1..*
1
1..*
1
*
*
1
2
2
1

5.2 Abstract and concrete

- the portType is the abstract interface definition
- *binding* makes it concrete — 'use HTTP'

  the same abstract interface may be available with various concrete bindings
- *port* describes where to find an implementation of the binding

  a single binding could be available at several locations

5.3 Example: Google Web API

- three methods: search, spell, fetch from cache
- return rich data structures with all the entry data seen on the search page
- think of an application!
- (now sadly obsolete)

5.4 GoogleSearch.wsdl

```
<definitions
  name="GoogleSearch"
  targetNamespace="urn:GoogleSearch"
  xmlns:typens="urn:GoogleSearch"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
```

[in the following, the sections are not presented in order]

5.5 PortType

- a number of operations, characterized by their messages
- messages may be named
- may optionally specify a number of fault messages, to be returned in place of second message
- (Google uses a generic SOAP Fault to report 'bad key' and 'max exceeded')

```
<portType name="GoogleSearchPort">
  <operation name="doGoogleSearch">
    <input message="typens:doGoogleSearch"/>
    <output message="typens:doGoogleSearchResponse"/>
  </operation>
</portType>
```

5.6 Operation types

- one-way (just `<input>`)
- request-response (`<input>`, `<output>`)
- notify (`<output>`)
- solicit-response (`<output>`, `<input>`)
- HTTP only supports first two naturally
- use higher-level constructs for more complex interactions, and to set up end-points for notification and solicitation

5.7 Message

- named sequence of parts, each with name and type
- `binding` element describes how these are mapped onto transport protocol representations

```
<message name="doGoogleSearch">
  <part name="key"            type="xsd:string"/>
  <part name="q"              type="xsd:string"/>
  <part name="start"          type="xsd:int"/>
  <part name="maxResults"     type="xsd:int"/>
  <part name="safeSearch"     type="xsd:boolean"/>
  ...
</message>

<message name="doGoogleSearchResponse">
  <part name="return"         type="typens:GoogleSearchResult"/>
</message>
```

5.8 Types

- XMLSchema provides the basic type system
- repeating groups must be modelled using the `array` data type of the SOAP encoding namespace

```
<types> <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
                    targetNamespace="urn:GoogleSearch">

  <xsd:complexType name="GoogleSearchResult">
    <xsd:all>
      <xsd:element name="estimatedTotalResultsCount" type="xsd:int"/>
      <xsd:element name="resultElements"
                   type="typens:ResultElementArray"/>
      ...
      <xsd:element name="searchTime" type="xsd:double"/>
    </xsd:all>
  </xsd:complexType>
```

```
<xsd:complexType name="ResultElement">
  <xsd:all>
    <xsd:element name="summary" type="xsd:string"/>
    <xsd:element name="URL" type="xsd:string"/>
    <xsd:element name="snippet" type="xsd:string"/>
    <xsd:element name="title" type="xsd:string"/>
    ...
  </xsd:all>
</xsd:complexType>
<xsd:complexType name="ResultElementArray">
  <xsd:complexContent>
    <xsd:restriction base="soapenc:Array">
      <xsd:attribute ref="soapenc:arrayType"
        wsdl:arrayType="typens:ResultElement[]"/>
    </xsd:restriction>
  </xsd:complexContent>
</xsd:complexType>
</xsd:schema> </types>
```

5.9 Binding

- how to format the message for a particular transport
- binding gets a unique name; type relates to portType
- convenient for re-use if just one portType and one binding
- `soap:binding` indicates SOAP over HTTP, and in `rpc` style rather than `document`
- could instead have SOAP over SMTP or over FTP
- SOAP action is carried over into an HTTP header; describes intent (use deprecated)
- input and output messages encoded in SOAP message body
- to use an SMTP binding, we would need to change the `transport` tag, maybe select document-centric processing, set the address to be a `mailto:` (see below), etc

```
<binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="doGoogleSearch">
    <soap:operation soapAction="urn:GoogleSearchAction"/>
    <input>
      <soap:body use="encoded"
        namespace="urn:GoogleSearch"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </input>
    <output>
      <soap:body use="encoded"
        namespace="urn:GoogleSearch"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
    </output>
  </operation>
</binding>
```

Document and RPC style, encoded and literal

- RPC-style binding

  message body contains one element, named after operation; parameters or return value as sub-elements of this wrapper

- document-style binding

  message body contains arbitrary XML document, not necessarily conforming to SOAP rules (the default)

- encoded appearance of message

  XML structure is ‘abstract’; encode as per `encodingStyle`

- literal appearance of message

  XML structure is ‘concrete’

- `document/encoded` is never used

- `use="encoded"` is deprecated by WS-I anyway

5.10 Port, Service

- gives the network address for the binding
- service could list several ports (e.g. one bound to HTTP and one bound to SMTP)

```
  <service name="GoogleSearchService">
    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
      <soap:address location="http://api.google.com/search/beta2"/>
    </port>
  </service>
</definitions>
```

5.11 WSDL: Other features

- any WSDL element can contain a `<documentation>` element, with free-form comments
- WSDL document can be split into parts, and an `<import>` element used to recombine; naturally, the included elements are pointed to by a URI
- a plain HTTP GET/POST binding is possible (so we could write WSDL that describes regular form processing)
- MIME can also be used

Approximately:

```
<binding name="SearchHTTPBinding" type="typens:GoogleSearchPort">
  <http:binding verb="GET"/>
  <operation name="doGoogleSearch">
    <http:operation location="search"/>
    <input>
      <http:urlEncoded/>
    </input>
    <output>
      <mime:content type="text/html"/>
    </output>
  </operation>
</binding>
```

5.12 Criticising WSDL

- rudimentary treatment of semantics
- separate documentation generally required
- low-level; use other languages for co-ordination, transactions, etc.
- multiple purposes (logical/architectural/operational)

  convenient for consumer to get all at once, but not for producer(s): WSDL should be a derived format

6 Universal Description, Discovery, and Integration (UDDI)

- centralized registry of services
- supporting a *marketplace*: providers, consumers, brokers
- manually searchable through web site: '*static web services*'
- programmatically searchable through API: '*dynamic web services*'

6.1 Registries

- major, public: `uddi.microsoft.com`, `uddi.ibm.com` etc, *mirrored*
- public test: `test.uddi.ibm.com`
- private — in a large organisation, for internal management
- semi-private — 'extranets' etc.

- operate as web servers, web services

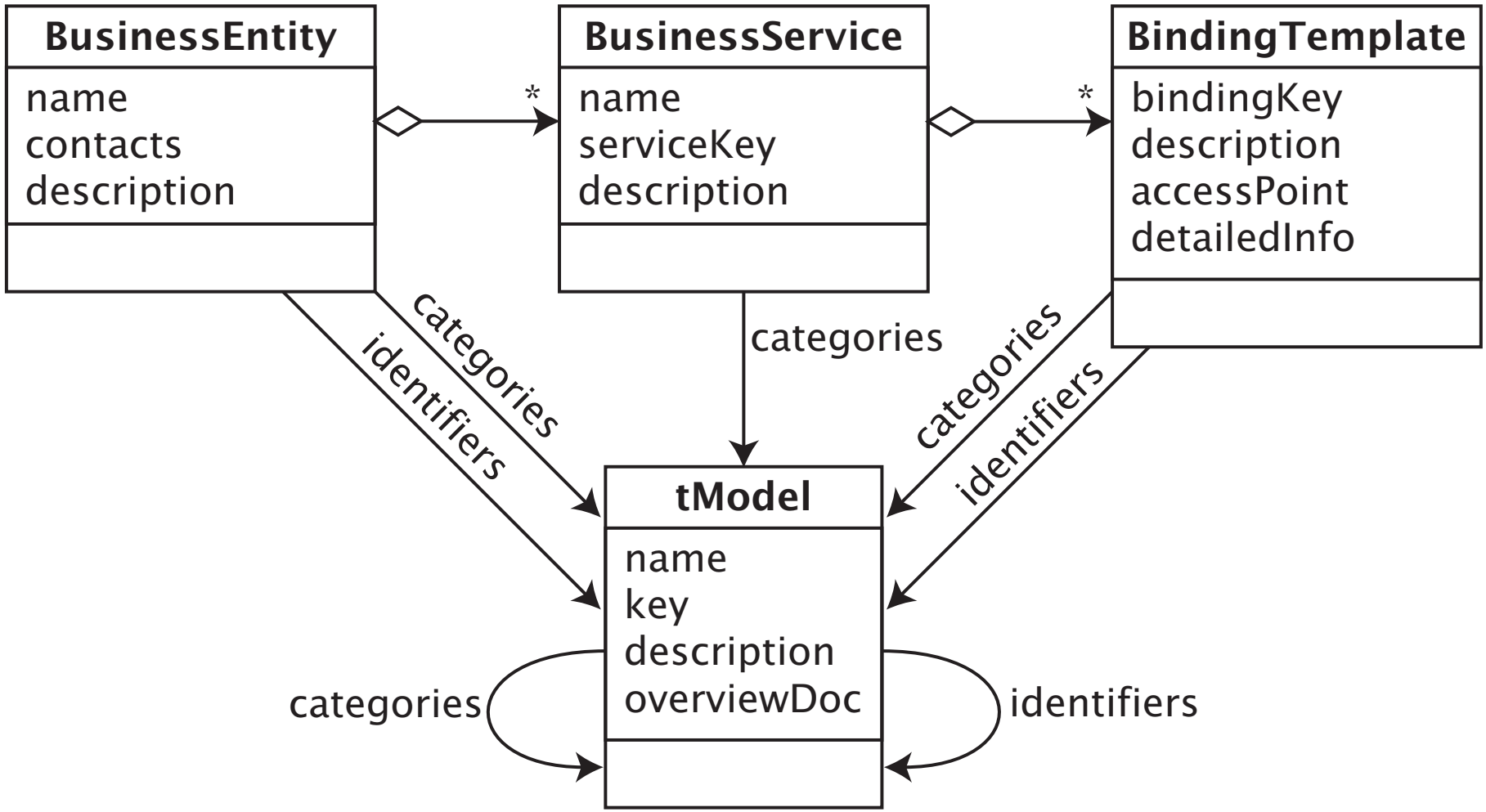
6.2 Registry terminology

- *business entity*: info about service provider; ‘white pages’
- *business service*: non-technical categorization of service provided; ‘yellow pages’
- *taxonomy*: business standard descriptions
- *publisher assertions*: business relationships
- *binding template*: technical service access information; often a URI, WSDL port; ‘green pages’
- *tModel (technical model)*: meta-data for service; could be text, WSDL document

6.3 tModels

- tModels are the key
- but are specified outside UDDI
- expected to be published by industry consortia
- programmers retrieve model description and implement a service which claims to conform
- detailed specifications stay with the owner, not in the registry

6.4 Registry structure



6.5 Searching

Methods to search on each kind of data:

find_business(), *find_service*(), *find_binding*(), *find_tModel*()

Methods to get the corresponding data:

get_businessDetail(), *get_serviceDetail*(), etc

6.6 Publishing

save_business(), *save_service*(), *delete_business*(), etc — usually requires authentication!

6.7 UDDI shortcomings

- UDDI Business Registry shut down in January 2006:

UDDI Business Registry

With the approval of UDDI v3.02 as an OASIS Standard in 2005, and the momentum UDDI has achieved in market adoption, IBM, Microsoft and SAP have evaluated the status of the UDDI Business Registry and determined that the goals for the project have been achieved. Given this, the UDDI Business Registry will be discontinued as of 12 January 2006.

For additional information, see the IBM UDDI Business Registry - Shutdown FAQ.

- mission accomplished!

Actually, it didn't work in practice:

> The idea was that companies could publish how they wanted to interact, and other companies could find that information and use it to establish a relationship. Needless to say, this isn't how companies do business – there's always a human element to establishing a relationship. As a result, the UBR served as little more than an interoperability reference implementation.
>
> (Jason Bloomberg, *ZapThink*, quoted in *InfoWorld* 2005-12-16)

> There was no quality enforcement or governance on the UBR, because there was no direct funding. Anyone could publish. No one was checking quality or doing garbage collection. So there was lots of, um, low-quality data in the UBR.
>
> (Dino Chiesa, *All About Interop*, 2005-12-16)

7 Web services in practice

- not just one standard, but many — 70+!
- three different standards organisations (W3C, OASIS, WS-I)
- even for the fundamental five, interoperability isn't automatic
- hence WS-I, 'the standards organization for standardizing the standards' (2002–)

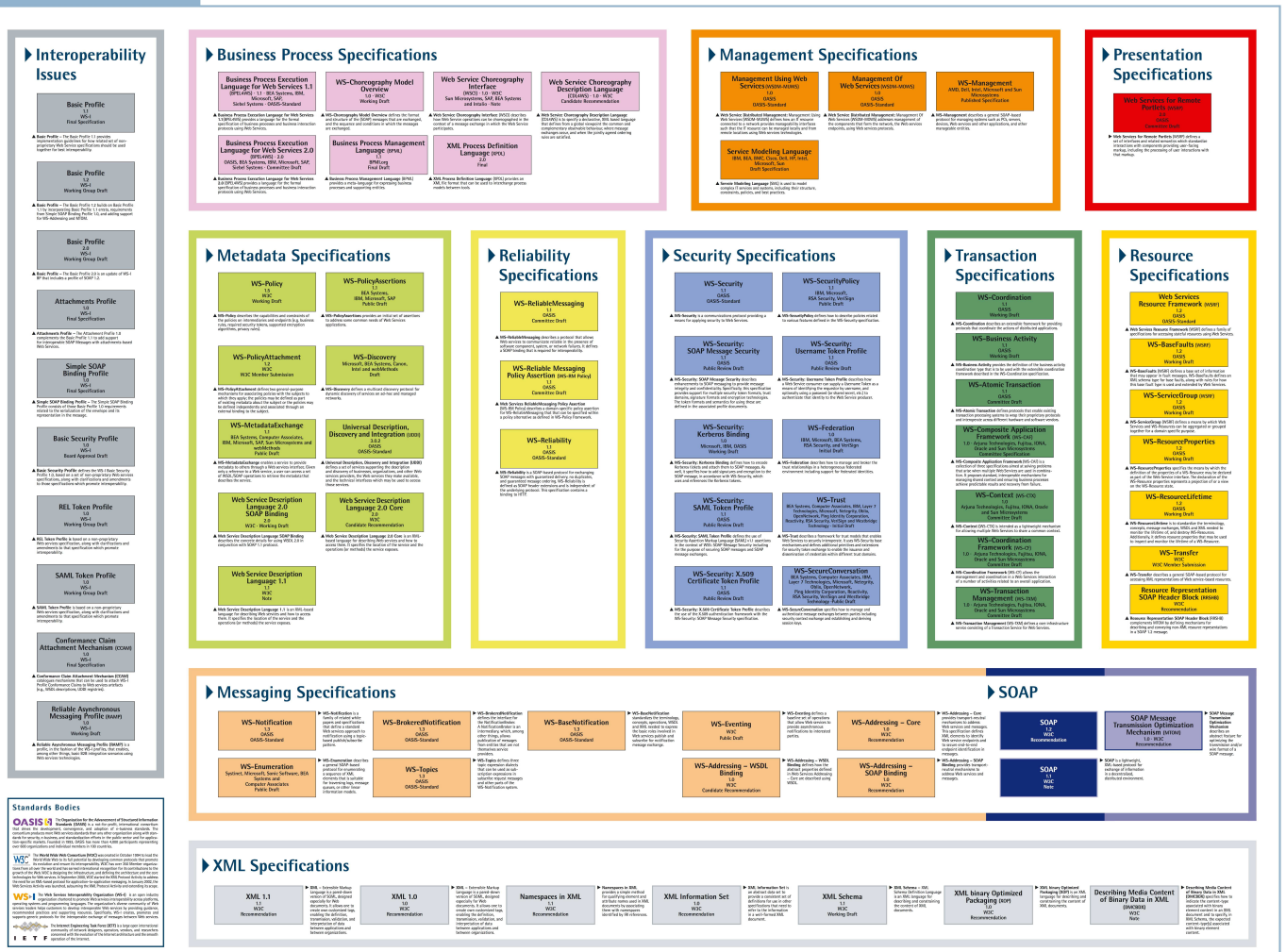
7.1 WS-I profiles

- specific versions of specific standards
- plus guidelines and conventions for interoperability
- WS-I Basic Profile 1.1 (2004): XML 1.0, WSDL 1.1, SOAP 1.1, HTTP 1.1, UDDI v2

  `http://www.ws-i.org/Profiles/BasicProfile-1.1.html`
- also WS-I Basic Security Profile 1.0 (2007)

  `http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html`
- interoperability still an issue outside these profiles

Web Services Standards Overview



Index

Contents

Service-Oriented Architecture

Monday	Tuesday	Wednesday	Thursday	Friday
Introduction Components	REST	Composition	Architecture	Engineering
coffee	coffee	coffee	coffee	coffee
Components	REST	Composition	Architecture	Conclusion
lunch	lunch	lunch	lunch	lunch
Web Services	Qualities	Objects	Semantic Web	
tea	tea	tea	tea	
Web Services	Qualities	Objects	Semantic Web	