Distributed Systems

Web Services

Dynamic and Distributed Information Systems

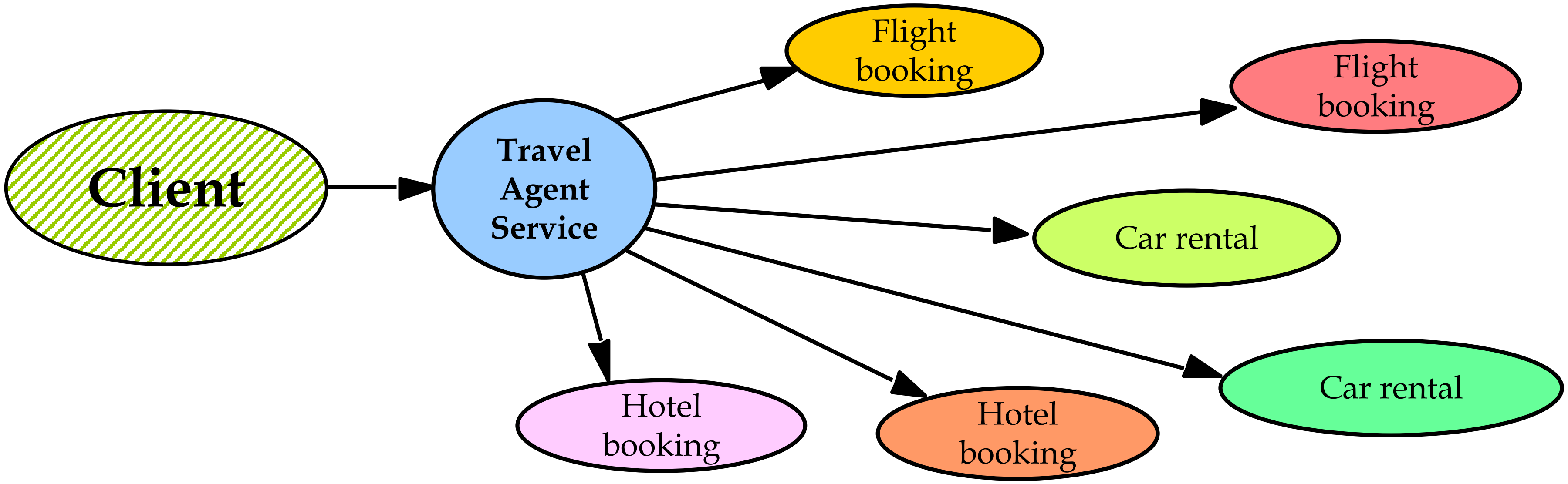
Today’s Agenda

- Architecture Overview of Web Services
- SOAP
- WSDL
- UDDI
- Comparison to CORBA

Architecture Overview

Dynamic and Distributed Information Systems

Example: Travel Agent



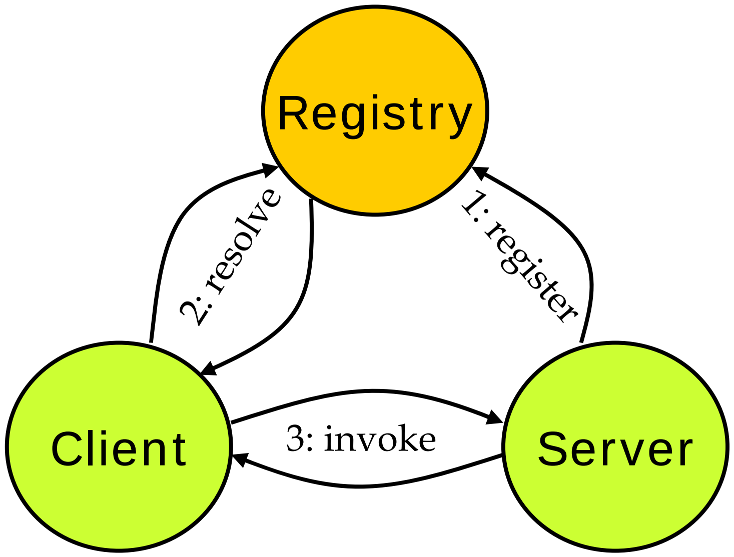
Travel agent scenario

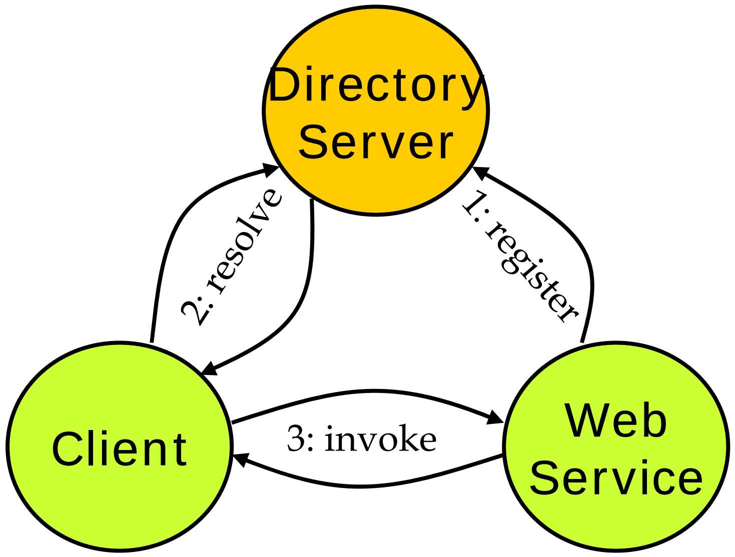
1. The client asks the travel agent service for information about a set of services; for example, flights, car hire and hotel bookings.
2. The travel agent service collects prices and availability information and sends it to the client, which chooses one of the following on behalf of the user:
   (a) refine the query, possibly involving more providers to get more information, then repeat step 2;
   (b) make reservations;
   (c) quit.
3. The client requests a reservation and the travel agent service checks availability.
4. Either all are available;
   or for services that are not available;
   either alternatives are offered to the client who goes back to step 3;
   or the client goes back to step 1.
5. Take deposit.
6. Give the client a reservation number as a confirmation.
7. During the period until the final payment, the client may modify or cancel reservations

General Architecture of Web Services

RPC/RMI Architecture

Web Services Architecture





Basic Points (1/2)

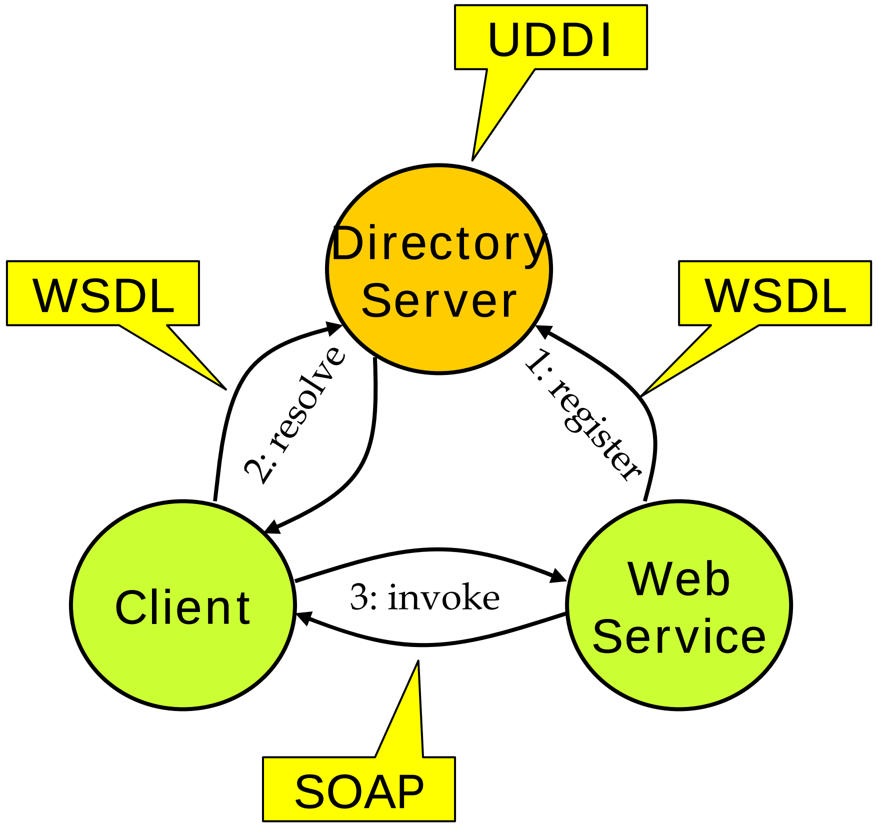
- Wire Protocol
  - Interaction between remote sites
  - Should work over any transport protocol (TCP/IP, HTTP, SMTP, etc.)
    - Therefore, should be based on messages (instead of procedure calling)
  - **SOAP**
    - Formerly: "Simple Object Access Protocol"
    - Now: just "SOAP" (more generic, not restricted to object access)
  - Common syntax for all specifications:
    - **XML**: used for all standards in Web Services
  - Defines standardized conventions that:
    - turn invocations to XML messages
    - exchange the message
    - turn the XML message to a service invocation

Basic Points (2/2)

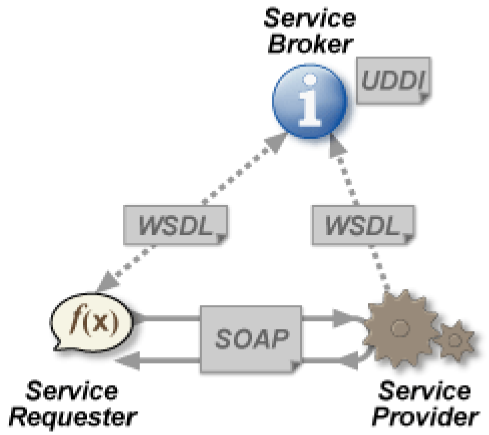
- Description of Web Services
  - Standardized way to describe service interfaces
  - **WSDL**: Web Service Description Language
    - XML-based Interface Definition Language (IDL)
    - "Hide" web service implementation behind an interface
    - Language independent

- Discovery of Web Services
  - **UDDI**: Universal Description, Discovery, and Integration
  - Like a registry or naming service of typical distributed systems

Web Services Architecture



The Web Services Stack



http://en.wikipedia.org/wiki/Web_service

Wire Protocols	Description	Discovery
SOAP Blocks	Agreements	
SOAP/XMLP	Process	
XML	WSDL Extensions	
HTTP/SMTP/BEEP	WSDL	Registry (UDDI)
TCP/IP	XML	Inspection

Wire Protocols: **SOAP**

Dynamic and Distributed Information Systems

Wire Protocols

- Primary Role:
  provide a standard, flexible communications channel
- Secondary Role:
  provide a standard, flexible wire-level data representation
- Advantage:
  interoperability at the lowest level

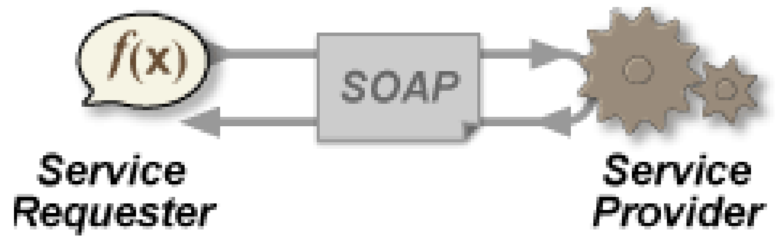
What SOAP specifies

- Defines a message format, encoded in XML
- Conventions for using SOAP messages to implement RPC-like interaction
  - Defines how a client can invoke a remote procedure by sending a SOAP message, and how the server can reply by sending another SOAP message back
- Rules about how to process a SOAP message, and exception handling in case some parts of a message are not understood by the recipient
- Description of how SOAP should be transferred on top of various transport layers, including HTTP (Hyper Text Transfer Protocol) and SMTP (Simple Mail Transfer Protocol)

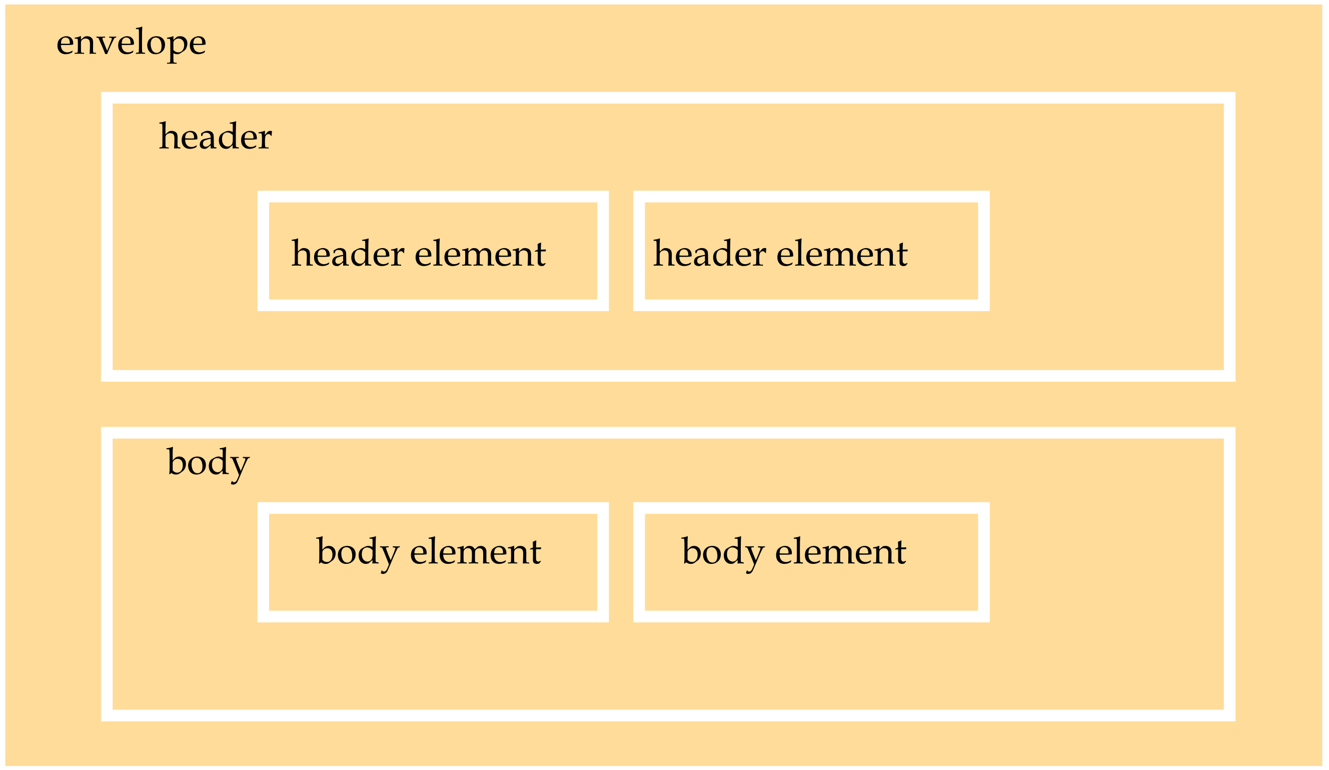
Example: Implementing RPC with SOAP

- A conventional RPC call
  - takes input arguments
  - invokes the required method on the server synchronously
  - returns output
- Steps to implement with SOAP
  - Encode input parameters and call to a procedure in a SOAP message
  - Encode response output in another SOAP message
  - To ensure synchronous invocation, use HTTP transport instead of SMTP
    - HTTP is synchronous: client sends an HTTP request, and get the reply in the synchronous HTTP response
    - SMTP is asynchronous: client sends SOAP message by email, and receives another email with the server’s reply later on, asynchronously

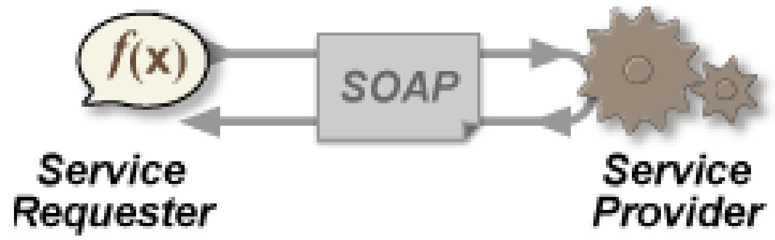
SOAP message format



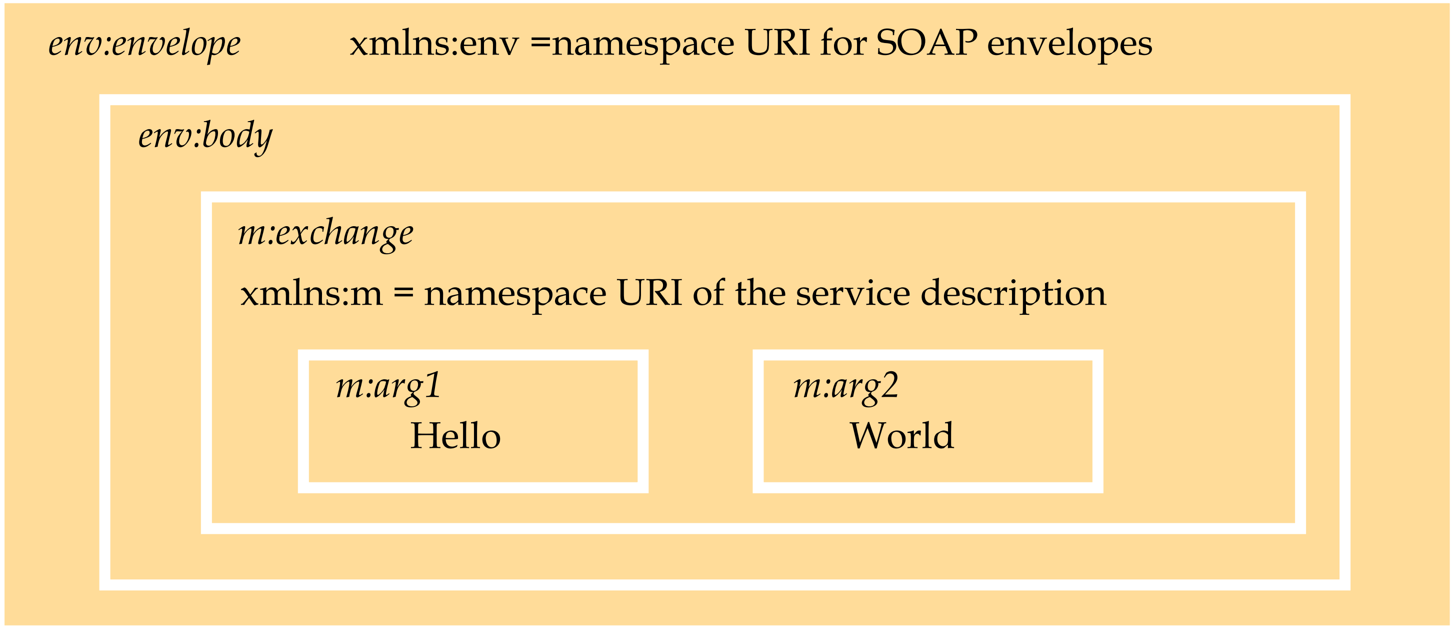
http://en.wikipedia.org/wiki/Web_service



SOAP: Example of a request



http://en.wikipedia.org/wiki/Web_service



env:envelope xmlns:env =namespace URI for SOAP envelopes

env:body

m:exchange

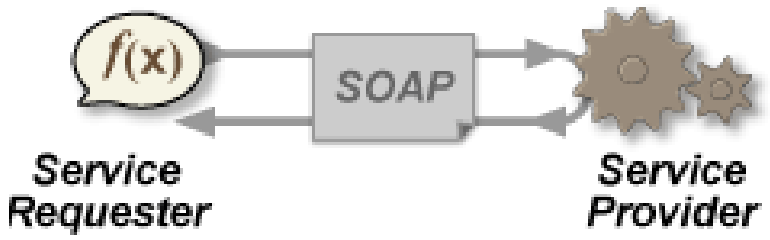
xmlns:m = namespace URI of the service description

m:arg1 Hello

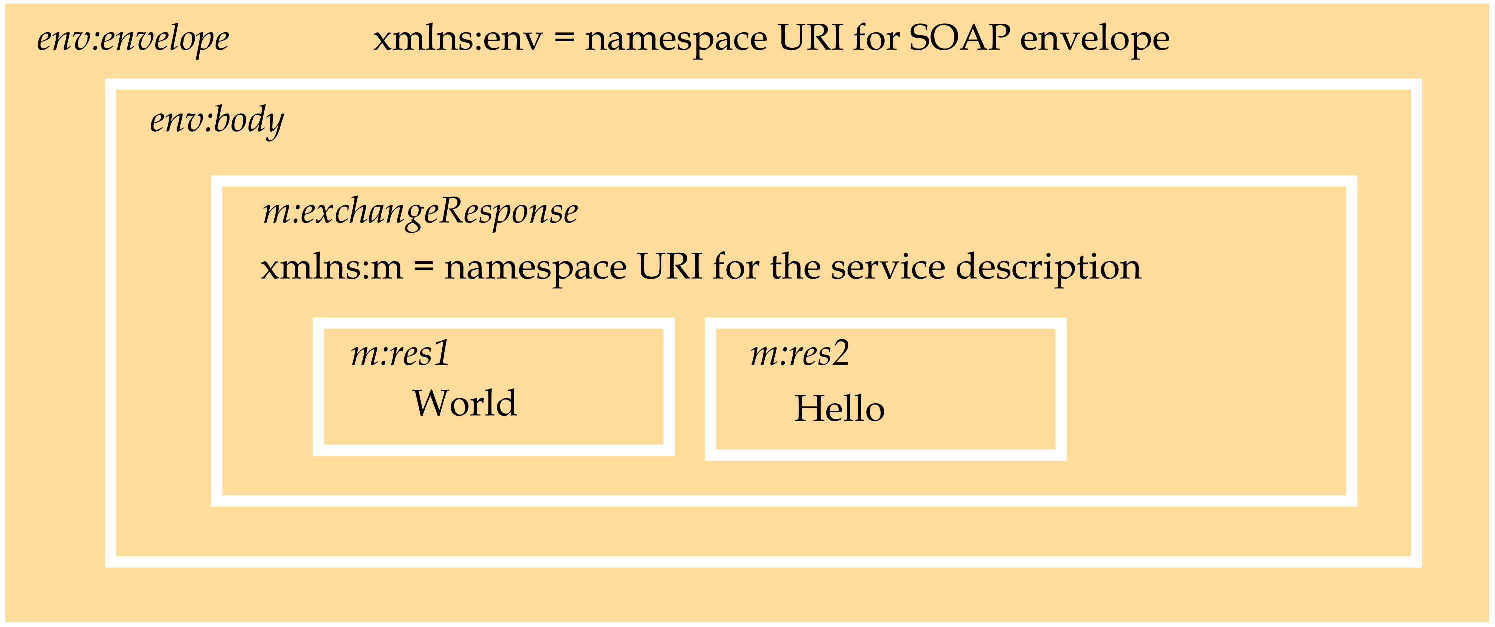
m:arg2 World

Each box represents an XML element with its name in italics followed by any attributes and its content

SOAP: Example of a reply



http://en.wikipedia.org/wiki/Web_service


env:envelope xmlns:env = namespace URI for SOAP envelope
env:body
m:exchangeResponse
xmlns:m = namespace URI for the service description
m:res1
World
m:res2
Hello

SOAP: Use of HTTP POST Request

```
POST /examples/stringer                                   ← endpoint address
Host: www.cdk4.net
Content-Type: application/soap+xml
Action: http://www.cdk4.net/examples/stringer#exchange    ← action

<env:envelope xmlns:env=    namespace URI for SOAP envelope   >
<env:header> </env:header>
<env:body> </env:body>
</env:Envelope>
```

HTTP header

SOAP message

SOAP: Request in XML

```
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetails
         xmlns="http://warehouse.example.com/ws">
        <productID>827635</productID>
    </getProductDetails>
  </soap:Body>
</soap:Envelope>
```

SOAP: Response in XML

```
<soap:Envelope
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <getProductDetailsResponse
         xmlns="http://warehouse.example.com/ws">
      <getProductDetailsResult>
        <productName>Toptimate 3-Piece Set</productName>
        <productID>827635</productID>
        <description>3-Piece luggage set. Black
                    Polyester.</description>
        <price>96.50</price>
        <inStock>true</inStock>
      </getProductDetailsResult>
    </getProductDetailsResponse>
  </soap:Body>
</soap:Envelope>
```

Description: **WSDL**

Description

- Primary Role:
  provide a standard, flexible way to describe what and how a Web service does what it does.

- Advantage:
  interoperability

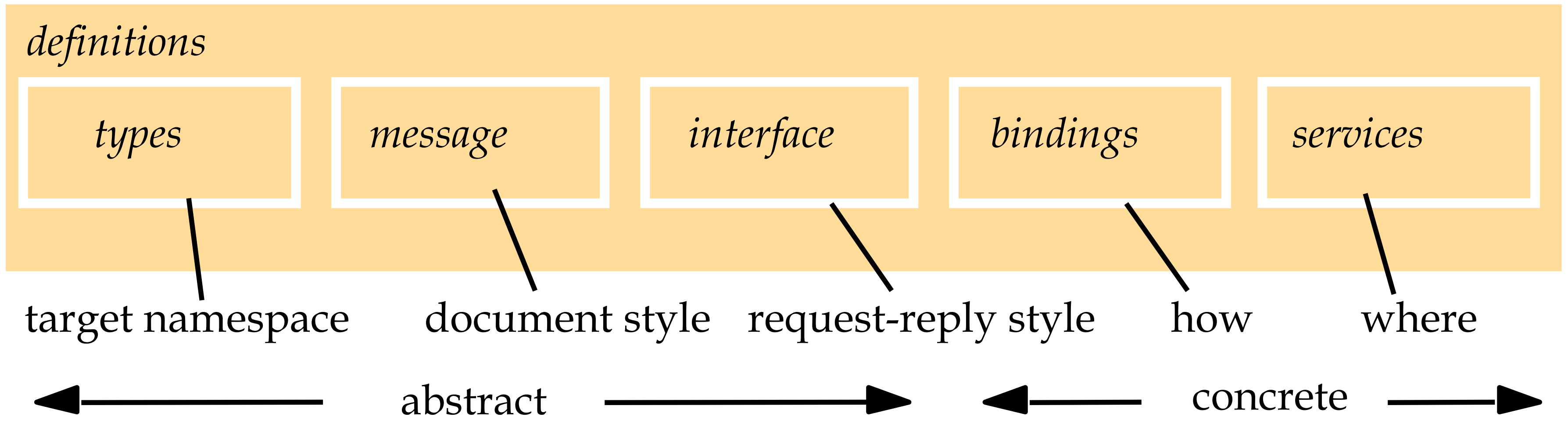
WSDL – Intro

- WSDL: Web Service Description Language
- XML syntax for formally describing how to invoke a web service
  - Which calls are accessible
  - Through which mechanism (SOAP or other)
  - What are the inputs and outputs
  - Where is the service located
  - What is the transport layer (HTTP or other)

WSDL – The Major points

- WSDL is a simple XML grammar for describing how to communicate with a Web service
  - It defines the messages (both abstract and concrete) that are sent to and from a service
  - It defines logical collections of messages (“port type”, “interface”)
  - It defines how a given “port type” is bound to particular wire protocols
  - It defines where the service is located

Main elements in WSDL description



WSDL – The Hairy Details

```
<definitions>
   <types>   <!-- XML Schema -->   </types>
   <message name="getQuote_In" />
   <message name="getQuote_Out" />
   <portType name="StockQuoteServiceInterface">
      <operation name="getQuote">
         <input message="getQuote_In" />
         <output message="getQuote_Out" />
      </operation>
   </portType>
   <binding name="StockQuoteServiceBinding" type="StockQuoteServiceInterface">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" />
      ...
   </binding>
   <service name="StockQuoteService">
      <port name="StockQuoteServicePort" binding="StockQuoteServiceBinding">
         <soap:address location="http://www.acme.com/services/stockquote" />
      </port>
   </service>
</definitions>
```

Definition of data types

Definition of messages

Definition of port type

Definition of the bindings

Definition of the service

WSDL – sum up

- WSDL is extensible.
- WSDL was created by IBM and Microsoft
  - The intent was to create something that worked, not something that was complete
  - Creating a formal Web Services "data model" was not a priority

Discovery: **UDDI**

Dynamic and Distributed Information Systems

Discovery

- Primary Role:
  provide a standard, flexible way to discover where a Web service is located and where to find more information about what the Web service does (the *description*)

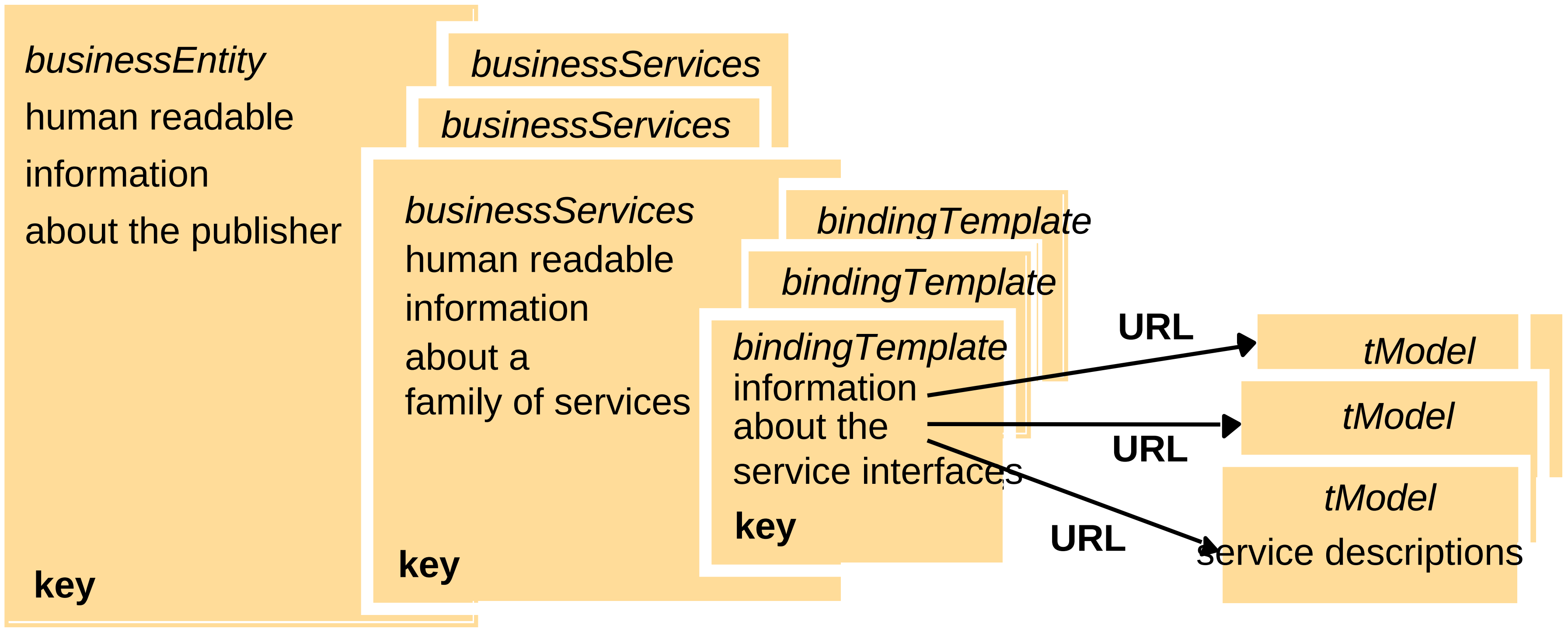
- Advantage:
  interoperability, dynamic integration

UDDI Overview

UDDI is:

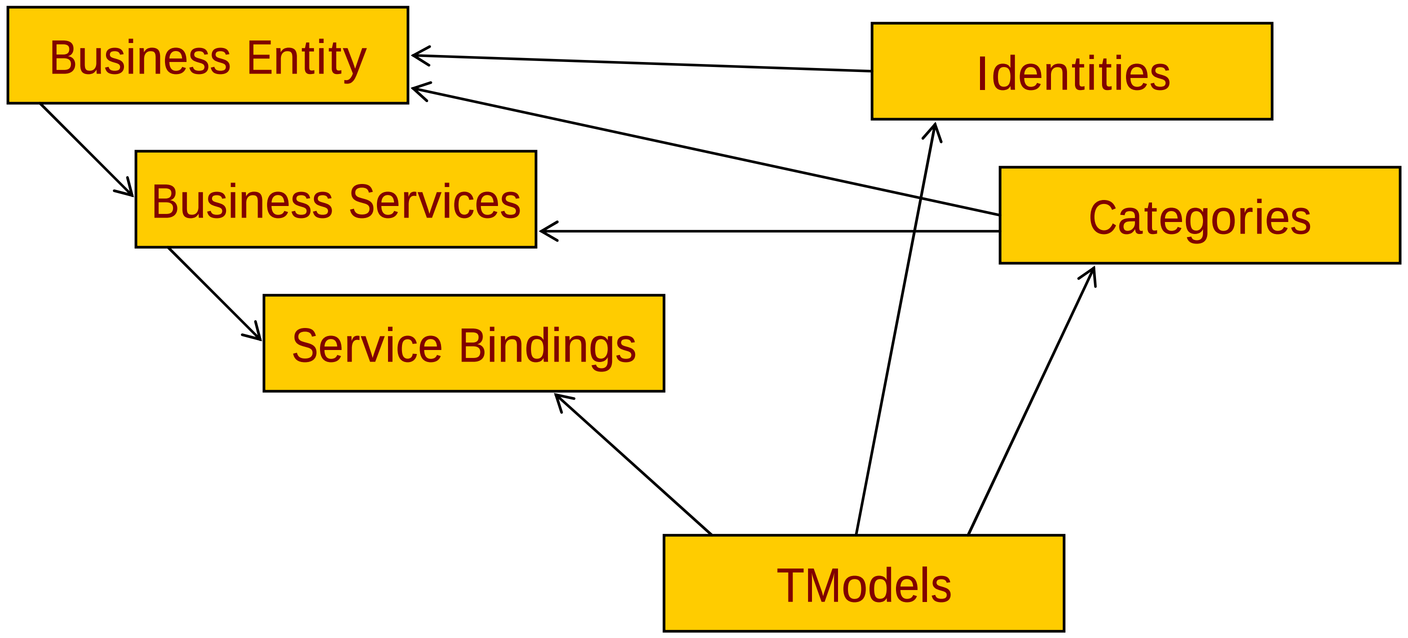
- A Web Services API for publishing and discovering the existence of Web services
- A registry for managing information about Web services
- A coalition of organizations working together to manage UDDI registries and to further develop the Web Services API for accessing those registries.

The UDDI data structures



Yellow Page model

- UDDI is built around a "Yellow-pages" like data model:



UDDI – TModels

- TModel = "Technology Model"

TModel

Abstract metadata definition relating to some aspect of the UDDI registration

TModel Instance

Implementation specific metadata conforming to a given TModel.

TModel = Abstract Class

UDDI – TModels

- TModels
  - Categories & Identifiers
    - Categorization and Identification taxonomies are TModels
    - Categories and Identifiers are TModel Instances
    - Keyed References
      - Name + Value + TModel
    - Examples: NAICS, UNSPSC, D&B #
  - WSDL Port Types
    - WSDL Port Types are TModels
    - WSDL Services that are bound to a Port Type are TModel Instances
  - WSFL Business Processes
    - WSFL Flow Models are TModels
    - WSFL Global Models are TModel instances

TModels represent the extent of UDDI’s semantic description capabilities.

UDDI - Conclusions

- UDDI has only limited extensibility through TModels
- UDDI was created by IBM, Microsoft and Ariba (many companies have joined the effort)
- The intent was to put something together that worked.

Comparison to CORBA

Dynamic and Distributed Information Systems

Web Services vs. CORBA

- CORBA is designed to run in an organization
  Web Services are designed to run on Internet
- Remote object references != URIs
  - In CORBA type identifiers refer to ORB-repository and aren't generally understood
- Naming
  - DNS/UDDI is loosely coupled
- HTTP/XML are simple, CORBA has learning curve
- Efficiency
  - XML isn't as efficient as CORBA with its binary formats
- CORBA has
  - Transactions, concurrency control, security, access control, persistent objects