Web Services Objective

- **Modular applications**
- **Self-contained, self-describing**
- **Published, located, and invoked across the Web**
- **Using standard, open protocols**
  - TCP/IP, XML, SOAP, UDDI, WSDL, …
- **Automation**
- **Different processes use same data in stand. way**
- **Companies create tighter relationships**
  - Trading partners, vendors, customers
  - E.g, Amazon apparel store
- **Exchange data more quickly**

Case Study: Amazon

- **Services Exported**
  - Product details (short, long, images, samples)
  - Purchase functionality
  - Ratings, reviews, collaborative filtering data, lists, …
- **Examples**
  - Store builder tools
  - Amazon Browser – visualization tool
  - Windows desktop interfaces – drag-n-drop…
  - MP3 Piranha
  - Games

Case Study: Google

- **Services Exported**
  - Search interface
  - Limits on items returned, queries / day
- **Examples**
  - Metacrawler functionality
  - Geosearch ‘nearby thai restaurants’
    - TIGER, FIPs -> lat,long of pages
  - Robust hyperlinks
    - Creates a signature for destination pages & tracks with query

Case Study: Hailstorm / MyServices

- **Web Services**
  - MyDocuments
  - MyAddressbook
  - MyWallet
  - MyNotifications
  - …
- **Scenario**
  - Wallet keeps receipts, arranges product return
  - Expedia uses notifications to warn of canceled flight
- **Reality**
  - Ebay, AmEx, Groove, …

Five Key Requirements

- Standard way to represent data
  - XML
- Common, extensible, message format
  - SOAP
- Common, extensible, service description language
  - WSDL
- Way to discover services on a particular Web site
  - DISCO
- A way to discover service providers
  - UDDI

What Is XML?

- **eXtensible Markup Language for data**
  - Standard for publishing and interchange
  - “Cleaner” SGML for the Internet
- **Applications:**
  - Data exchange over intranets, between companies
  - E-business
  - Native file formats (Word, SVG)
  - Publishing of data
  - Storage format for irregular data
  - …

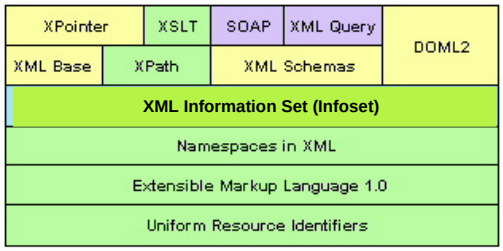
What's Special about XML?

- **Simple, Supported**
- **Easy to parse**
  - Even with no info about the document
- **Can encode data**
  - With little, or
  - With much structure
- **Data references inside & outside document**
- **Programmatic Interfaces**
  - SAX – streaming interface, forward, read-only cursor
  - DOM – tree-oriented operations, in memory, expensive
- **Many, many tools**

Important XML Standards

- XSL/XSLT*:
  - presentation and transformation standards
- RDF:
  - resource description framework (meta-info such as ratings, categorizations, etc.)
- Xpath/Xpointer/Xlink*:
  - standard for linking to documents and elements within
- Namespaces:
  - for resolving name clashes
- DOM:
  - Document Object Model for manipulating XML documents
- SAX:
  - Simple API for XML parsing

Tower of Standards



Basic XML Structures

Elements:
Open & close tags or "empty tag"
Ordered, nestable

Attributes:
Single-valued, unordered
Special types:
ID, IDREF, IDREFS

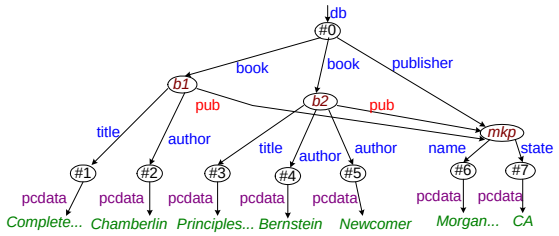
PCDATA/CDATA

```
<?xml version="1.0" encoding="UTF-8"?>
<db>
  <book ID="b1" pub="mkp">
    <title>Complete Guide to DB2</title>
    <author>Chamberlin</author>
  </book>
  <book ID="b2" pub="mkp">
    <title>Transaction Processing</title>
    <author>Bernstein</author>
    <author>Newcomer</author>
  </book>
  <publisher ID="mkp">
    <name>Morgan Kaufman</name>
    <state>CA</state>
  </publisher>
</db>
```


In Contrast to HTML

- **XML must be well formed**
- **No HTML dangling <p>**
- **Strict hierarchical containment**
  - No foo
- **Can view as a tree**
  - IDREFs (specified in DTD) create graph structure
  - But there is still a unique, tree-based parse.

Graphical View



Other XML Structures

- **Processing instructions: instructions for applications**
  `<?xml version="1.0"?>`
- **CDATA sections: treat content as char data**
  `<![CDATA[<tag>Whatever!!!</tag><whatever>]]>`
- **Comments: just like HTML**
  `<!-- Comments -->`
- **Entities: external resources and macros**
  - `&my-entity;` (non-parameter entity)
  - `%param-entity;` (parameter entity for DTD declarations)

Document Type Descriptor

- **Inherited from SGML DTD standard**
- **BNF grammar**
- **Constraints on element structure and content**
- **Specification of attributes and their types**
- **Definitions of entities**

Example DTD

Means optional

```
<!ELEMENT paper(author*, date, abstract?, body>
  <!ATTLIST paper keywords CDATA #IMPLIED>
<!ELEMENT author(affiliation?, email, pcmember?)>
  <!ATTLIST author name CDATA #REQUIRED>
  <!ELEMENT affiliation (#PCDATA)>
  <!ELEMENT email (#PCDATA)>
  <!ELEMENT pcmember EMPTY>
<!ELEMENT abstract (p*|#PCDATA)>
<!ELEMENT body (section*)>
  <!ELEMENT section (heading, (p|fig|section)*)>
    <!ELEMENT p ((b|ref|#PCDATA)*)>
    <!ELEMENT b (#PCDATA)>
    <!ELEMENT ref (#PCDATA)>
      <!ATTLIST ref name IDREF #REQUIRED>
    <!ELEMENT fig (#PCDATA)>
      <!ATTLIST fig caption CDATA #IMPLIED>
```

Note IDREF

Two ways to specify a DTD

- **External DTD**

```
<?xml version="1.0"?>
<!DOCTYPE greeting SYSTEM "hello.dtd">
<greeting>Hello, world!</greeting>
```

- **Internal**

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
      <!ELEMENT greeting (#PCDATA)>
      ]>
<greeting>Hello, world!</greeting>
```


Shortcomings of DTDs

Useful for documents, but not so good for data:

- **No support for structural re-use**
  - Object-oriented-like structures aren't supported
- **No support for data types**
  - Can't do data validation
- **Can have a *single* key item (ID), but:**
  - No support for multi-attribute keys
  - No support for foreign keys (references to other keys)
  - No constraints on IDREFs (reference *only* a Section)

XML Schema

- **In XML format**
- **Includes primitive data types**
  - (integers, strings, dates, etc.)
- **Supports value-based constraints**
  - (integers > 100)
- **User-definable structured types**
- **Inheritance**
  - (extension or restriction)
- **Foreign keys**
- **Element-type reference constraints**

Sample XML Schema

```
<schema version="1.0"
   xmlns="http://www.w3.org/1999/XMLSchema">
<element name="author" type="string" />
<element name="date" type = "date" />
<element name="abstract">
  <type>
    …
  </type>
</element>
<element name="paper">
  <type>
    <attribute name="keywords" type="string"/>
    <element ref="author" minOccurs="0" maxOccurs="*" />
    <element ref="date" />
    <element ref="abstract" minOccurs="0" maxOccurs="1" />
    <element ref="body" />
  </type>
</element>
</schema>
```


Subtyping in XML Schema

```
<schema version="1.0"
   xmlns="http://www.w3.org/1999/XMLSchema">
<type name="person">
  <attribute name="ssn">
  <element name="title" minOccurs="0" maxOccurs="1" />
  <element name="surname" />
  <element name="forename" minOccurs="0" maxOccurs="*" />
</type>
<type name="extended" source="person"
   derivedBy="extension">
  <element name="generation" minOccurs="0" />
</type>
<type name="notitle" source="person"
   derivedBy="restriction">
  <element name="title" maxOccurs="0" />
</type>
<key name="personKey">
  <selector>.//person[@ssn]</selector>
  <field>@ssn</field>
</key>
</schema>
```

XML Namespaces

- **Motivation**
  - An XML markup vocabulary may be used in >1 doc
- **Namespace = collection of names**
  - Each identified by an URI
  - Used as element types and attribute names
- **Two namespaces are identical if URIs are string=**
- **Declare namespace using reserved attr xmlns:***
- **Scoping, defaulting**

```
<?xml version="1.0"?>
 <!-- unprefixed element types are from "books" -->
 <book xmlns='urn:loc.gov:books'
       xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <title>Cheaper by the Dozen</title>
    <isbn:number>1568491379</isbn:number>
 </book>
```


Web Services

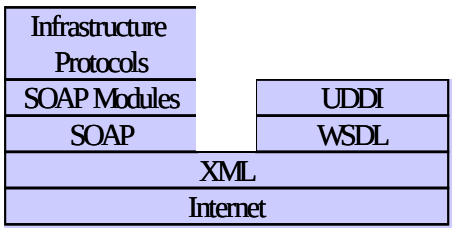
- ✓ **XML**
- ✓ **Schemas**
- ✓ **Namespaces**

- **Route Messages**
  - SOAP
- **Transform XML content**
  - XSLT
- **Locate Services**
  - WSDL
  - UDDI

SOAP

- **Remote Procedure Call**
- **Three Parts**
  - Envelope (what's in message and how to process)
  - Optional header (datatype encoding rules)
  - Convention for representing RPCs and responses
- **TCP/IP, HTTP not *required***

Distributed Messaging



SOAP Modules

- **WS-Security**
  - Credential exchange, message integrity, confidentiality
- **WS-License**
  - Describes common license types
    - X.509 certificates and Kerberos tickets
  - How to place them in WS-Security credentials tag
- **WS-Routing**
  - Deal with proxy serves / load balancers
  - One-way, 2-way, peer-to-peer, long dialogs
- **WS-Referral**
  - Dynamic configuration of the routing path
- **WS-Inspection**
  - Find services. Ties WSDL and Disco together

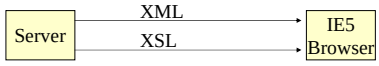
XSL

Two languages in one

- **Allows transformation of XML to**
  - XML
  - HTML
  - WAP
  - …..
- **Also allows formatting of XML**
  - HTML
  - PDF
  - ...

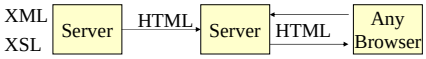
XSL Processing

- **On the client**



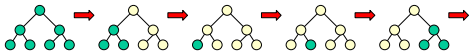
- **Dynamically, on the server**

- **Preprocessing, on the server**



XSL Processing

- **XSL maps from XML tree to a new tree**
- **XSL document contains list of template rules**
  - rule = match pattern, template content
- **XSL processor scans thru input tree**
  - Typically, looking at each subtree in turn: apply-templates
  - DFS order



- **When pattern matches a subtree**
  - Output template content
- **PCDATA output as default**

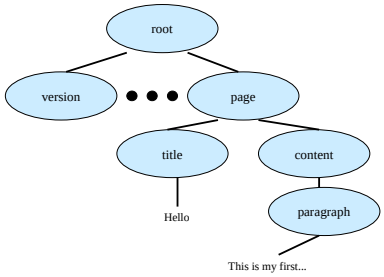
Hello XML Example

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>

<!-- Written by Stefano Mazzocchi "stefano@apache.org" -->

<page>
 <title>Hello</title>
 <content>
  <paragraph>This is my first Cocoon file!</paragraph>
 </content>
</page>
```


Graphical Model



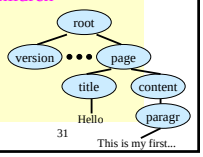
XSL Transform

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
 <xsl:template match="page">
  <html>
    <head>
      <title> <xsl:value-of select="title"/> </title>
    </head>
    <body bgcolor="#ffffff">
      <xsl:apply-templates/>
    </body>
  </html>
 </xsl:template>
 <xsl:template match="title">
  <h1 align="center">
    <xsl:apply-templates/>
  </h1>
 </xsl:template>
 <xsl:template match="paragraph">
  <p align="center">
    <i> <xsl:apply-templates/> </i>
  </p>
 </xsl:template>
</xsl:stylesheet>
```

Namespace

Compute value of title node and output that

Then recursively process children



Browser Sees...

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.0//EN"
"http://www.w3.org/TR/RED-html40/strict.dtd">
<html>
 <head><title>Hello</title></head>
 <body bgcolor="#ffffff">
   <h1 align="center">Hello</h1>
   <p align="center">
     <i>This is my first Cocoon file!</i>
   </p>
 </body>
</html>
<!-- This page was served in 93 milliseconds by Cocoon 1.5 -->
```

From template

Default

XSL elements, attributes

apply-templates	**E**	**Process children**
value-of	**E**	**Compute value of node**
for-each	**E**	**Iterate over**
select	**A**	**Which nodes (used in all above)**

Template Matching

```
<xsl:template match="/"
   <html> ... </html>
```

Just match root

```
<xsl:template match="album"
   ...
```

Match album elements

```
<xsl:template match="album/track"
   ...
```

Match specific parent/child


```
<xsl:template match="library//name"
   ...
```

Match all "name" descendants of library

Match by ID, match attributes @, comments, processing instructions…
Use logical or |, perform tests []

Expressions for Selecting Nodes

- **Value of select attribute is XPath expression**
- **Xpath expressions are superset of match patterns**
- **Hierarchical composition: axis::expression**
- **Some example axis**
  - child, ancestor, descendant, following, preceding-sibling

```
<xsl:template match="award">
  <p><xsl:value-of select="parent::actor"/></p>
</xsl:template>
```

- **XPath is simple XML Query Language**

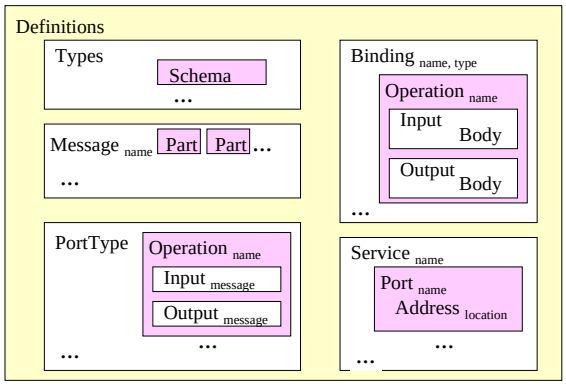
If a WS were a Phone Call…

- **XML**
  - represents the conversation,
- **SOAP**
  - describes the rules for how to call someone
- **UDDI**
  - is the phone book.
- **WSDL**
  - describes what the phone call is about and how you can participate.

WSDL

- **Abstract Definitions**
  - **Types**:
    - Machine- and language-independent type definitions.
  - **Messages**:
    - Input, output parameters (name, type)
  - **PortTypes**:
    - Operation signatures (name, input+output parameters)
- **Concrete Descriptions**
  - **Bindings**:
    - One for each operation in the PortTypes section
  - **Services**:
    - Specifies port addresses of each binding

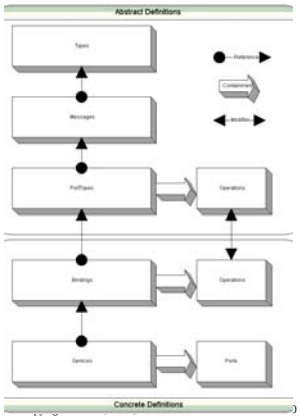
WSDL Structure



int foo(int arg);

```
<types>
 <schema targetNamespace="http://tempuri.org/xsd"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas...l/" elementFormDefault="qualified" >
 </schema>
</types>
<message name="Simple.foo">
  <part name="arg" type="xsd:int"/>
</message>
<message name="Simple.fooResponse">
  <part name="result" type="xsd:int"/>
</message>
<portType name="SimplePortType">
  <operation name="foo" parameterOrder="arg" >
    <input message="wsdlns:Simple.foo"/>
    <output message="wsdlns:Simple.fooResponse"/>
  </operation>
</portType>
```


WSDL



DISCO

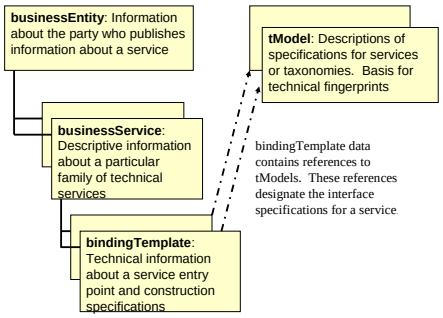
- **If you know the URL for a service**
- **DISCO lets you query them**
- **And get back a WSDL description**

- **But what if you don't know the right URL?**

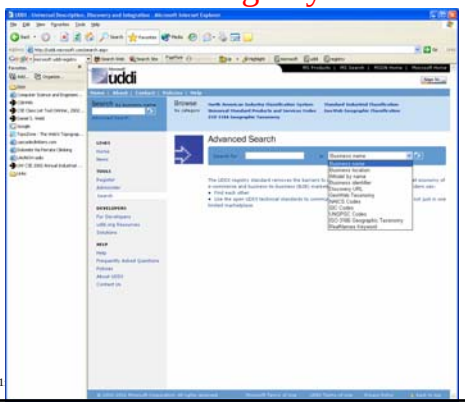
UDDI

- **Hosted Registries**
  - Microsoft, IBM, HP, SAP, NTT, BEA
- **Entries defined with**
  - Business information
    - Name, contacts, descriptions, identifier, yellow pages category
  - Service information
    - Entities, each of which describes a family of related services which together implement a business process
  - Binding information
    - How to invoke: URI, required parameters, options, & Tmodel
  - Service specifications (Tmodel)
    - As a symbol – fingerprint to recognize a known service
    - Decomposable to find WSDL description

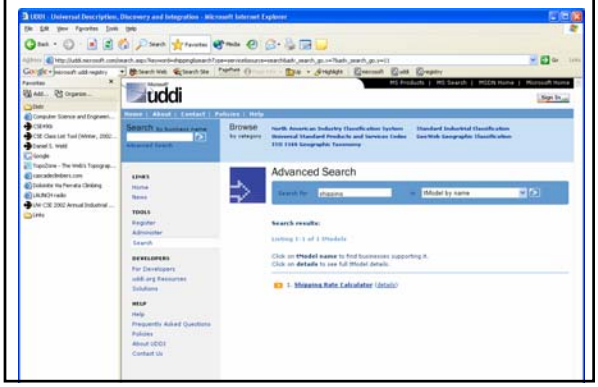
UDDI Components



What's in a Registry?

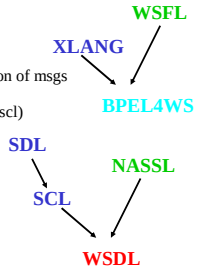


Search for "Shipping" in Tmodel



Acronyms (W3C, MSFT, IBM)

- **UDDI**
  - Discover, describe, register services
  - SOAP-based service for locating WSDL-formatted service descriptions
- **DISCO**
  - Discover / retrieve SCL+SDL descrips
- **SDL / NASSL**
  - SOAP description lang –get params / types
- **SCL**
  - SOAP contract lang – extends SDL – orchestration of msgs
- **WSDL**
  - Network services as endpoints on msgs (extends scl)
  - Uniform way of describing abstract interface and protocol bindings of arbitrary network services
- **XLANG / WSFL / BPEL4WS**
  - lang for biz processes used in BizTalk
  - Biz process execution language for web services
    - MSFT, IBM, BEA proposal



BPEL4WS Example