Table of Contents

GOALS

Purpose

The purpose of this course is to introduce you to XML, its associated technologies, and practical development practices so you can make informed technology choices and begin creating your own applications.

Training Objectives

- To understand current and emerging XML standards, including DTD, Schema, XSLT, XPath and XLink
- To recognize the benefits and drawbacks of XML versus HTML, SGML, EDI, and other standards.
- To review sample code for producing XML, consuming XML, and displaying XML.
- To review sample applications to give you a head start in your development.

INTRODUCTION

Description

HTML:

```
<HTML>
<HEAD><TITLE>Catalog Item</TITLE></HEAD>
<body>
<H3>The XML Handbook</H3>
<B>Sale price: $24.95</B>
<I>(Suggested retail: $39.95)</I>
<P>Shipping cost: $4.00 UPS Ground</P>
</body>
</html>
```

XML:

```
<?xml version="1.0"?>
<CatalogItem>
<Description>The XML Handbook</Description>
<Price type="sale" currency="USD">24.95</Price>
<Price type="retail" currency="USD">39.95</Price>
<Shipping type="UPS Ground" currency="USD">4.00</Shipping>
</CatalogItem>
```

- Documents and data that are tagged for content, meaning, or use
- Self-describing data and documents
- Common syntax for expressing **structure** in documents and data
- Standard for exchanging data as well as documents
- Extensible system for defining communication languages

HTML:

```
<HTML>
<HEAD><TITLE>Romeo and Juliet</TITLE></HEAD>
<body>
<H1>Romeo and Juliet</H1>
<H2>William Shakespeare</H2>
<H3>Act 1</H3>
<H4>Scene 1.1</H4>
<P>Two households, both alike in dignity,...
   A pair of star-crossed lovers take their lives</P>
<P><i>Romeo:</i>...</P>
</body>
</html>
```

XML:

```
<?xml version="1.0"?>
<Play>
<Title>Romeo and Juliet</Title>
<Author>William Shakespeare</Author>
<Act ID="Act1">
   <Scene id="Scene1.1">
      <Prolog> Two households, both alike in dignity,...
            A pair of star-crossed lovers take their lives
      </Prolog>
      <Speech role="Romeo">...
   </Scene>
</Act>
</Play>
```

Standards

XML for content and metadata

DTD or Schema for document structure and constraints

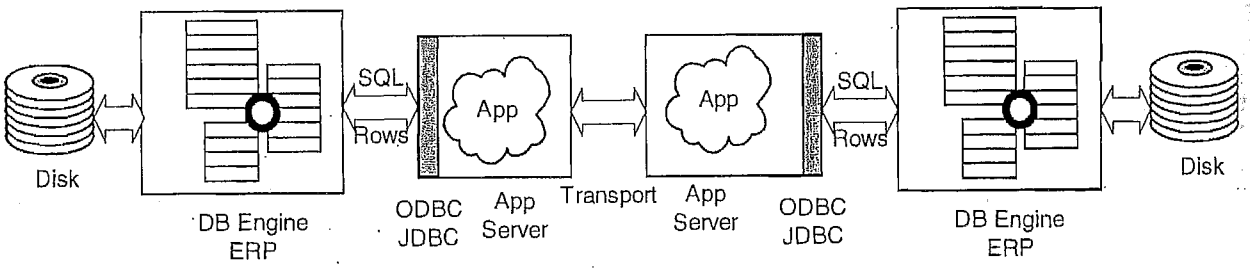
CSS and XSL FO for detailed display formatting

XSLT for layout on a page and transforming trees

XPath for identifying nodes in a tree (c.f. regular expressions or SQL)

XLink and XPointer for references between documents

Server Applications



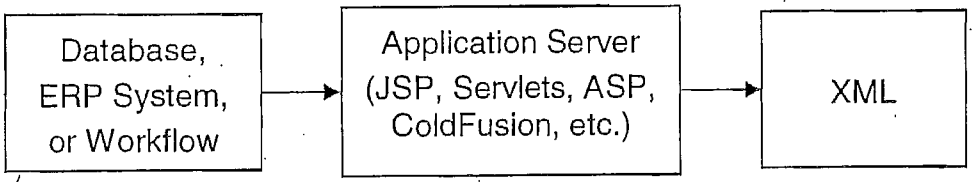
Benefits of XML

① Self-Describing

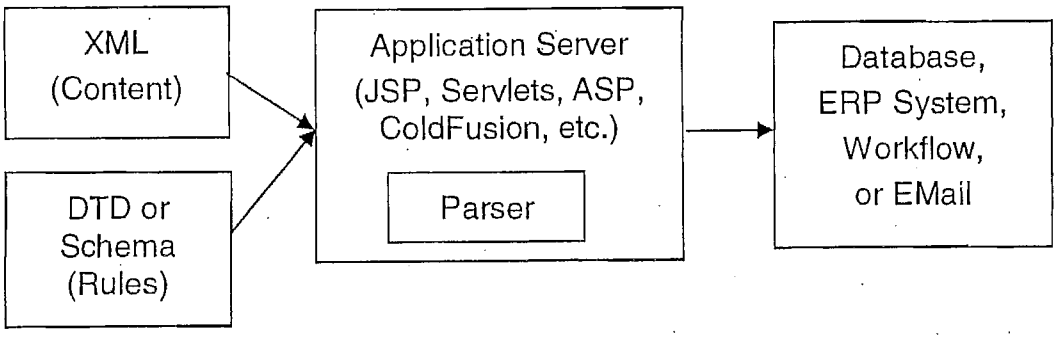
② Extensible
Extensible Markup Language

③ Generic Parser component free!!
Java - Xerces - xml.apache.org
C++ ...
MS (IE5+) MSDN.microsoft.com → MSXML (or) XML core services
.NET - wide bunch of XML capabilities

Producing XML:

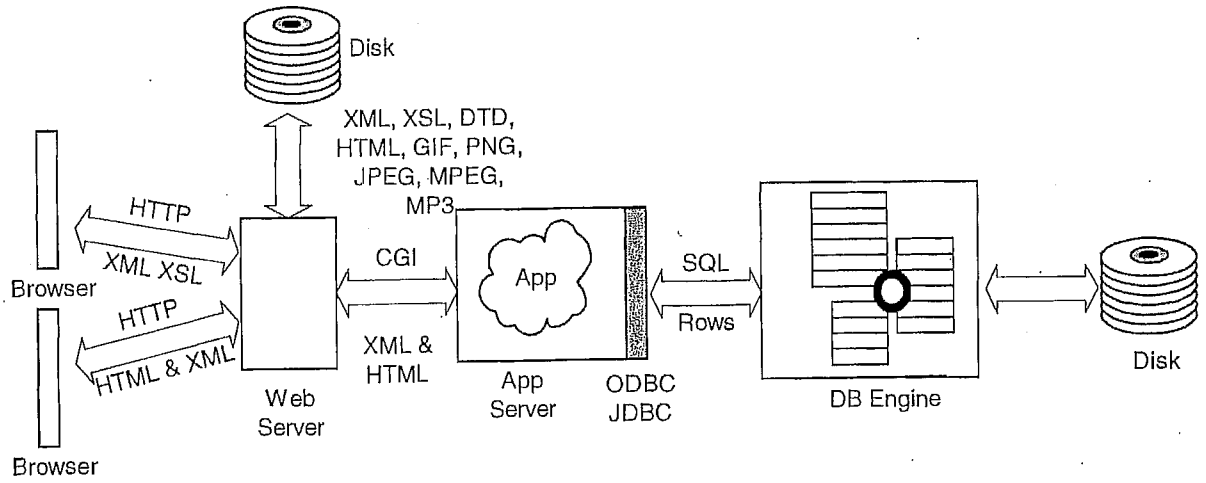


Consuming XML:

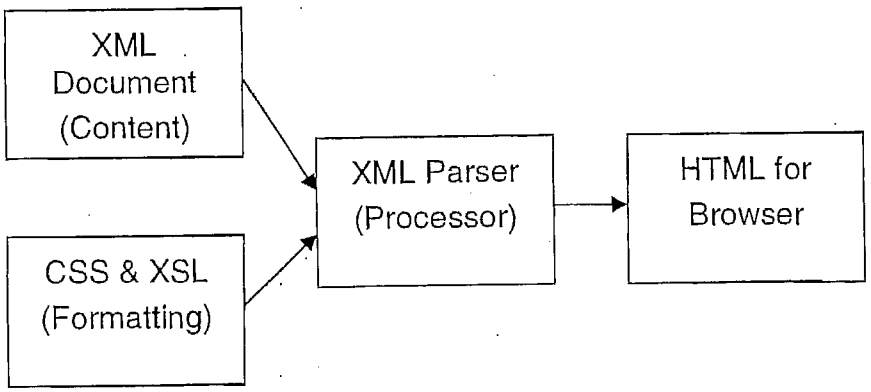


ERP - Enterprise Resource Planning

Client Applications



Within browser:



+ Smarter client—search, sort, filter on client to improve performance and reduce load on server

- Must have version 5 IE or Navigator 6 browser

Benefits

- Server to Server Communication: Allow industries to define platform-independent protocols for the exchange of data, especially the data of electronic commerce
- Smarter Clients: Deliver information to browsers in a form that allows automatic processing after receipt AJAX
- Repositories: Support both Presentation-oriented Publishing and Message-Oriented Middleware
- * Allow people to display information in a variety of renditions. By separating structure and content from presentation, the same XML source document can be written once, then displayed in a variety of ways
- Enable internationalized media-independent electronic publishing
- Enhanced searching and communication: Provide metadata -- data about information -- that will help people find information and help information producers and consumers find each other
- Support both human readers with text-based metadata, as well as automated machine processing with a strict hierarchical tree structure. Valuable corporate asset—data—represented in a simple, human readable form, as well as an easily processed form
- Databases are highly structured, while documents are unstructured. Using XML for the documents and adding a DTD or a schema and constraints makes them structured. They become as valuable a data or knowledge repository as databases.

* Style sheets can help you differentiate content format types in CSS which applies to the Browser the user/device is using

Uses HTTP.USER.AGENT

This way you can optimize style sheets in order to produce output meaningful – browser, device, printer, etc. even VoxML – voice

You can even make language adjustments – useful for small variance like driving instructions

You separate data from formatting

Comparison to SGML

The Standardized Generalized Markup Language, **SGML,** created and refined by Charles Goldfarb over a 20 year period. It became an international standard in 1986. All the major document/data concepts are already apparent in SGML:

+ Common document representation

+ Customized document types with domain-specific types

+ Represent structure rather than formatting—generalized markup

+ use stylesheets for particular renditions

+ make document abstraction descriptive enough so potential uses like searching, rendering particular formats, cataloging, etc. can be automated—treat documents more like data

+ Use rule-based markup, formally describing the rules in a Document Type Definition—the schema for documents

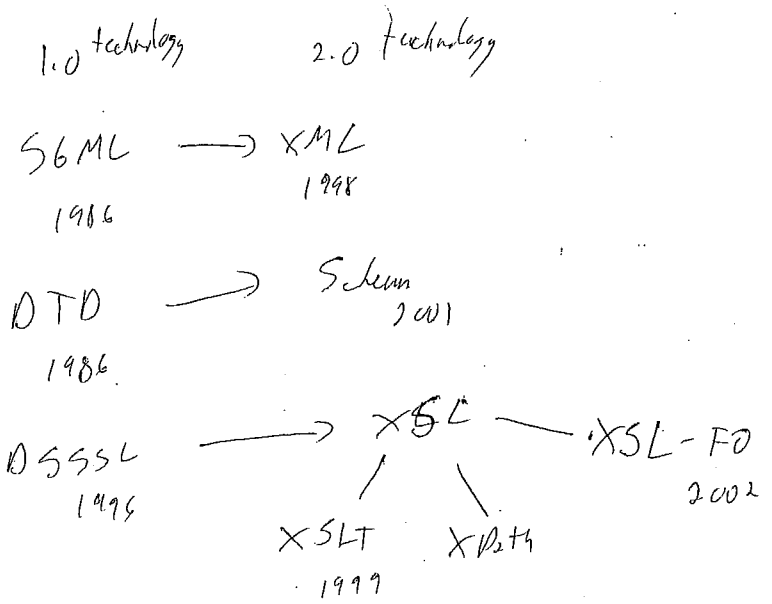
--Major drawback of SGML was expense in implementation.

XML is a simplified subset of SGML

XML is data oriented, rather than publishing oriented

XML is oriented towards short life span documents rather than massive document repositories

XML is much less flexible and has almost no optional features



Comparison to HTML

The HyperText Markup Language, **HTML,** is a specific application of SGML—one specific set of tags meant for linked documents.

It wasn't standardized until years after it was invented (1992), so it is not too rigorous about syntax. This makes browsers hard to write and maintain.

HTML is tough to extend. During the browser wars each vendor added their own extensions because standards bodies were too slow.

Formatting was creeping into HTML <b>, <i>, <font>, etc. until adoption of Cascading Style Sheets. CSS separated formatting from content so the formatting can be shared across many pages.

XML development started in 1996 and the core standard was issued in February 1998. The design goals were

1. XML shall be straightforwardly usable over the Internet.
2. XML shall support a wide variety of applications.
3. XML shall be compatible with SGML.
4. It shall be easy to write programs that process XML documents.
5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.
6. XML documents should be human-legible and reasonably clear.
7. The XML design should be prepared quickly.
8. The design of XML shall be formal and concise.
9. XML documents shall be easy to create.
10. Terseness in XML markup is of minimal importance.

meaning - overhead is in by description - SGML evolved

#1 drawback - XML is verbose

HTML made the Web the world's library. Information Superhighway

XML will make the Web the world's commercial and financial hub. Data Superhighway

The revolutionary benefit of the Internet is connectivity, but the connectivity does not necessarily imply understanding. HTML is a page layout language, not a language for communicating data.

HTML describes how your data should look, but XML describes what it means. A Web site can viewed as a fancy fax machine where what you see is all you get.

HTML is a very slowly changing set of tags with no structure or validation rules:

```
<H3>Sale price: $24.95</H3>
<I>(Suggested retail: $39.95)</I>
<B>Shipping cost: $4.00 UPS Ground</B>
```

XML is an extensible set of tags describing the structure of the document:

```
<PRICE type="sale" currency="USD">24.95</PRICE>
<PRICE type="retail" currency="USD">39.95</PRICE>
<SHIPPING type="UPS Ground" currency="USD">4..00</SHIPPING>
```

XHTML is a reformulation of HTML 4.0 in XML. There are two major reasons for content developers to adopt XHTML (from the spec):

"First, XHTML is designed to be extensible. This extensibility relies upon the XML requirement that documents be well-formed. Under SGML, the addition of a new group of elements would mean alteration of the entire DTD. In an XML-based DTD, all that is required is that the new set of elements be internally consistent and well-formed to be added to an existing DTD. The greatly eases the development and integration of new collections of elements.

Second, XHTML is designed for portability. There will be increasing use of non-desktop user agents to access Internet documents. Some estimates indicate that by the year 2002, 75% of Internet document viewing will be carried out on these alternate platforms. In most cases these platforms will not have the computing power of a desktop platform, and will not be designed to accommodate ill-formed HTML as current user agents tend to do. Indeed if these user agents do not receive well-formed XHTML, they may simply not display the document."

Comparison to EDI

Electronic Data Interchange

EDI defines standard data formats and transaction sets for electronic communication between trading partners. The benefits of EDI are

- faster turnaround on orders,
- better inventory control,
- complete real-time order and inventory information to improve planning and decision making,
- reduced costs compared to manual data entry systems.

However EDI also has problems:

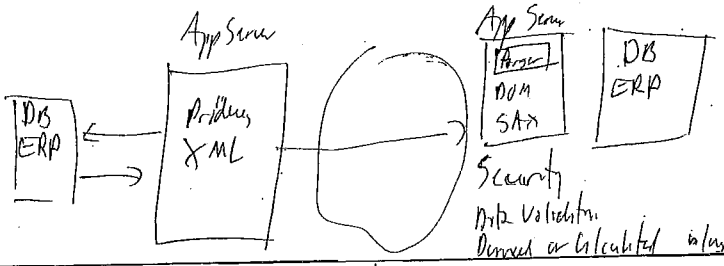
- Fixed transaction sets with business rules or process built into transaction sets
- Slow standards evolution
- High fixed cost, but variable benefits
- Limited adoption requires companies to preserve manual systems

XML is simpler, less expensive to implement, and more flexible (no predefined transactions or business rules) than EDI.

However XML is only a message format and does not supply all the other infrastructure of EDI. The other benefits of EDI are often available through other Internet standards:

- validation using validating parsers with DTDs or Schema.
- security with SSL
- transactional integrity with queued transactions and transaction monitors
- connection stability with QoS guarantees
- authentication with electronic signatures or digital certificates, etc..

Network services companies can provide validation, logging, audit trails, accountability, transaction roll-back, security, stable network infrastructure, and customer support.



www.xmlsoftware.com - list of XML tools (Rt sc 20/1564

Tools

Editing and Composition

Authoring tools make it easy to produce XML documents that adhere to a specific structure (DTD or Schema). These include FrameMaker, Adept.Editor, XMetaL, Microsoft Office 2000.

Other specialized editors simplify work with document type definitions (DTD), with Schema documents, and with stylesheets (CSS and XSL).

Content Management

You need a content management tool for applications where you store a large number of static XML documents. It provides a repository, versioning, access control, ad-hoc searching, deployment assistance, and group authoring support. Some sample tools are Astoria, POET, Texcel, Documentum, and many other content management solutions

Middle-tier

Producing XML from database content can be done with any application server technology. For example, Active Server Pages, Java Server Pages, servlets, ColdFusion, HoTMetaL APPS, etc. Specialized XML servers reduce the amount of code required.

Communicating XML can be done with a variety of transport protocols, such as HTTP, SSL, MQSeries, MSMQ, or even email. SOAP is a special protocol for distributed objects built on HTTP.

Consuming XML and acting on the content of a message starts with a parser (IBM & Sun Java parsers, Microsoft XMLDOM, and many others). Then code in an application server or other development environment acts on the results by updating a database, sending an email, charging a credit card, etc.

The new goal is platform independence using

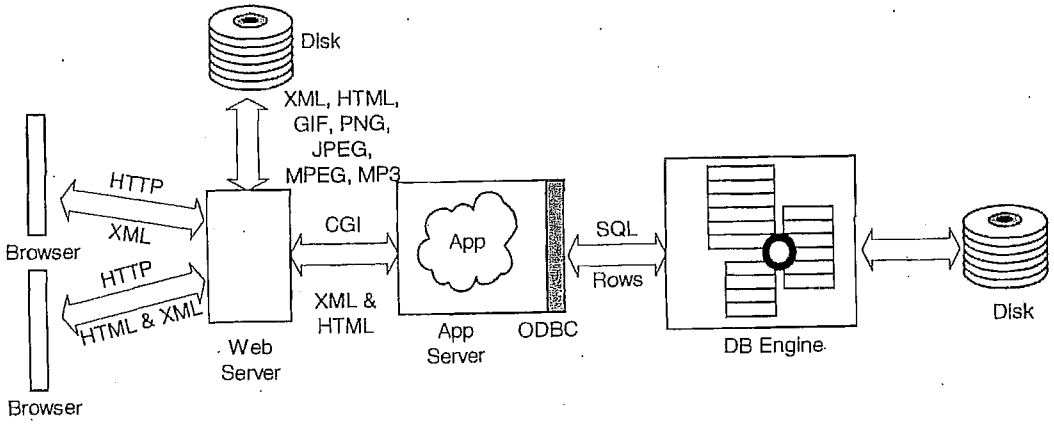
- Internet protocols for communication,
- XML for data format, and
- Java for programs.

Clients

Manipulate and display XML received. Sample clients include IE 5, Navigator 6, future printers, and voice synthesizers for the blind.

SAMPLE APPLICATIONS

Client Applications



Better searching on the Web using context or metadata—e.g. comparison shopping or authors of documents as opposed to commentators.

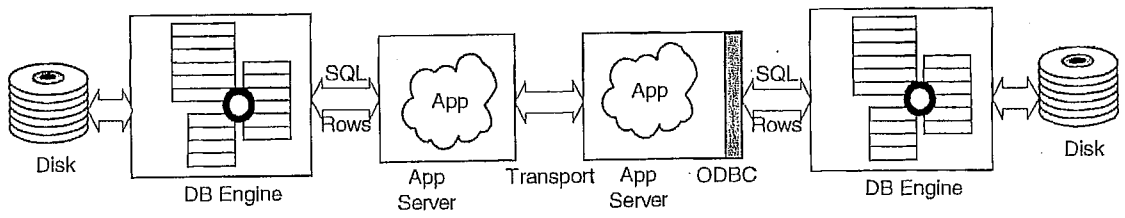
Move processing to client machine-querying, sorting, detail views, etc. Reduce network usage and improve client response times.

Multiple renditions on different output devices

Simplify internationalization by converting long documents into components that can be independently versioned and reused

Customized page layout pulling information from many servers

Server Applications



Integrating corporate systems—standard for communication between disparate corporate systems.

Supply chain integration—communication between trading partners

Validate documents in security regulation filings on Edgar.

Machine-readable specifications on all product descriptions

Accept and validate orders, invoices , and other business documents across the Internet

Integrate applications from across the company, e.g. accounting, human resources, manufacturing resource planning, etc.

Combine information from many databases

Financial wire transfers, telecommunications service provisioning, or medial record applications.

XML can integrate data from multiple sources for use in a single application.

Example: With commonly defined XML tags, a parts catalog application can enable a purchasing agent to examine the current price and availability of a needed parts component from each of the firm's approved suppliers to get the best deal, without having to understand the details of each supplier's inventory system.

XML can enable automatic application to application interchange.

Example: Once transaction information in agreed upon, an automotive part could be ordered automatically from the supplier, the supplier can verify the purchaser's trading terms, bill the invoice, and initiate the shipping request to the stockroom.

XML can be used to deliver customized data to different types of client devices.

Example: When XML tags are used to define the data in a customer database, a mobile salesperson could download a customer's account information and history into a Notes client on his/her laptop in order to review a new purchase, while another salesperson could obtain just the contact names and phone numbers in a personal organizer application.

Or, in the previous parts example, the purchasing agent could see the invoice on a Notes client, while the factory technician might see only the parts availability date on a PDA.

The XML metadata capabilities can help personalize the end user experience.

Example: XML enables the attachment of customer history and buying patterns to a purchase order, so as a user is navigating a Web site and browsing for more things to buy, the merchant can suggest other purchase options that would be consistent with previous buying habits.

XML can define the content of information delivered on the Web

Example: With XML-defined data embedded in Web pages, end users performing searches on the Internet will receive meaningful results in the context of their area of interest. For example, a book seller searching for books on automobiles will be able to search specifically for automobile titles without having to weed through search results from the many automobile sites that might reference a particular car.

Types of Documents

Content Documents

Here XML is used as a structuring mechanism for a body of documents to make them uniform and accessible. The development process is much like developing an interactive Web site with a focus on XML for content, custom DTDs to define the document structure, a document management system to manage the development and deployment, and XSL stylesheets for display.

Typical roles are a Producer, Information Designer, Layout Designer, Authors, and possibly Developers.

Business Documents

Here XML is used as a communication mechanism for documents that are accessible to both humans and automated processing. Examples are purchase orders, invoices, travel reimbursement, bill of lading, and other documents with specific roles in a business process.

XML helps automate the workflow through the business process, helps in the exchange of documents through the Internet between organizations, and simplifies the process of inserting or extracting data from database systems.

In developing such a system you typically adopt existing DTDs from your communication partners, analyze the business process and create a workflow to match, set up security and approval routing, design stylesheets for display, annd develop applications to integrate with databases.

Typical roles are a Standards Bearer for DTD definitions, Business Analyst for business processes, Architect for overall system design, Developers to implement the applications, User Interface Designer for stylesheets, and a Data Integrator to work with multiple corporate systems.

Protocol Documents

Here XML is used as a data encoding for communication between machines. Examples are distribution of weather observations, encoding of GPS readings during field surveys, distribution of software components, or remote control of other computers. XML helps formally define the message format (DTD) and simplifies encoding and decoding the messages.

In developing such a system you typically adopt existing DTDs from the applications with whom you would like to exchange data. You might design new DTDs for new application areas. Then you create the encoding and decoding software for both ends of the communication, possibly with translators to the application native formats.

You will need either a Standards Bearer for DTD definitions or a protocol designer for DTDs. Applications Developers implement the encoding, decoding, and interfaces to applications.

Corporate Applications

Information Distribution

XML plus XSL is a much more efficient than HTML for downloading data and allowing manipulation on the client, independent of the server. Separate the content from the display so the user can view the information in many different ways, possibly even personalizing its display. Distribute information rather than pages.

Knowledge Management

XML provides a structuring mechanism for reports, spreadsheets, emails, and memos to make them more accessible. A full knowledge management system goes further and actively encourages employees to record their experience to enable sharing and knowledge transfer. XML provides an excellent structured document format for modeling knowledge, leveraging existing documents, and providing context-sensitive searches.

Workflow

Using XML as the document format in a workflow modeling a business process simplifies the integration with existing systems, and through stylesheets makes the documents accessible from any browser. At each step along a workflow, an individual adds information to a document. A repository saves the final result, or possibly an audit trail from every significant checkpoint.

Application Integration

Corporations have often purchased best of breed applications for specific solutions in different parts of the organization. XML provides a *lingua franca* for communication between disparate applications. XSLT simplifies transformations to many different formats.

DOCUMENT CONTENT WITH XML

A **well-formed** document meets XML requirements on syntax. The basic rules for a well-formed document are:

- It consists of a simple tree structure with only one root element
- Elements nest in a clean tree structure without overlaps
- All tags have end tags or explicitly indicate there is no content
- Start and end tags have the same case
- Attributes for elements are name-value pairs with the value in quotes

These are much stricter than HTML. Here is an HTML example from Tim Bray:

```
<TITLE>Reasonable HTML</title>
Some text, and I <I>really</I> don't want a line break
between "line" and "break."
<P>Here's a picture: <IMG src=architier.jpg>
```

If we fix it up to be well-formed XML:

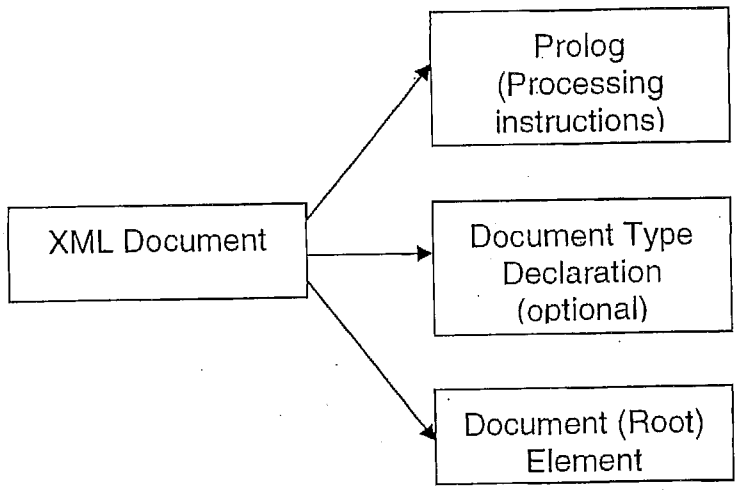
```
<HTML>
<TITLE>Well-formed XML</TITLE>
Some text, and I <I>really</I> don't want a line break
between "line" and "break."
<P>Here's a picture: <IMG src="architier.jpg"/></P>
</HTML>
```

One step beyond well-formed documents are **valid** documents. **Valid** documents conform to a particular standard, as expressed in a DTD or Schema, i.e. their structure and attributes match the requirements defined in the DTD.

Logical Document Structure

The prolog contains the XML declaration and (optional) document type declaration. It is followed by the document instance

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
...tree of elements
</memo>
```



Document element is root of a tree of elements. The structure of the tree is controlled by the DTD (optional).

Nodes
O-Elements
Δ - Attributes
□ - Text or CDATA character data

Example

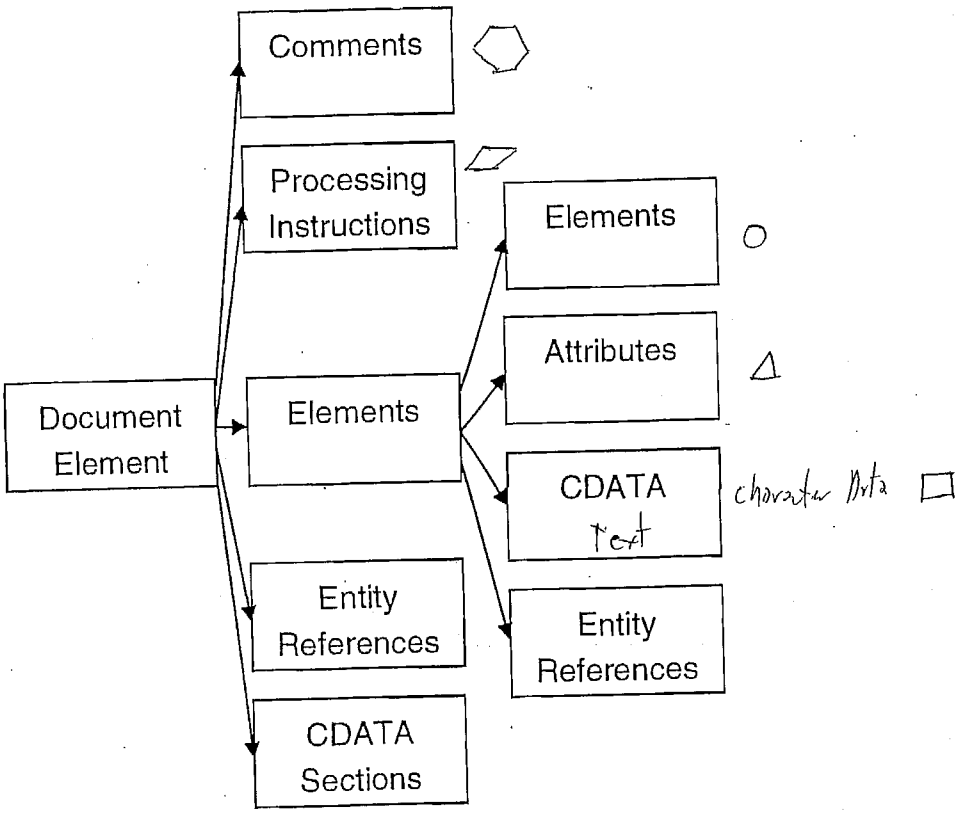
memo.xml

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
<From>
<Name>Ike van Cruyningen</Name>
<EMail>ikevc@architier.com</EMail>
</From>
<To Priority="1">XML Class</To>
<To Priority="2">My Coordinator</To>
<DateSent>2002-11-16</DateSent>
<Message Language="English">Does it all make
sense?</Message>
</memo>
```

□ - Text node holds the value.

Text node of "DateSent" is "2002-11-16"

Using DOM will require you to go one deeper than the element - to the CDATA



XML Syntax

Content

The actual information, i.e. content for people or data for databases, is known as character data (CDATA). Content is encased in markup.

Using the Unicode character set ensures easy internationalization. XML supports hex and decimal specification of Unicode (ISO 10646) characters with &#x*nnn*; and &#*nnn*;

Markup

Markup is the part of the document understood by the XML processor—the tags, entity references, and declarations. They all start with < or &.

The types of markup are:

- Elements
- Attributes
- Comments
- Entity references
- Processing instructions
- Document type declarations

White space (space, tab, carriage return, line feed) is ignored in markup, but preserved in character data.

Element, attribute, and other names start with a letter or underscore and can contain letters, digits, hyphens, underscores, colons, or periods. A name token, nmtoken, is restricted to the same set of characters and may start with letters, digits, hyphens, underscores, colons, or periods.

Literal strings are used for values and are surrounded by single or double quotes. If you need a quote within a string, use the other one as the delimiter. You may also use the character entities ' and " for quotes within a literal.

Elements

Elements represent the logical structure of a document. They are logical components, rather than physical components. Element type (or tag) names are case-sensitive, though you (or the DTD designer) get to choose the case.

```
<ElementTypeName attribute="value">
   child elements or content (CDATA)
</ElementTypeName>
<ElementTypeName attribute="value" />
```

Content may contain character data or other elements

```
<?xml version="1.0"?>
<CatalogItem>
<Description>The XML Handbook</Description>
<Price type="sale" unit="USD">24.95</Price>
<Price type="retail" unit="USD">39.95</Price>
<Shipping type="UPS Ground" unit="USD">4.00</Shipping>
</CatalogItem>
```

Comments use the same syntax as HTML: <!-- comment -->

If you do not want content to be interpreted as markup, put it inside a CDATA section:

```
<![CDATA[content which should not be parsed]]>
```

gives you ability to use special characters in content

2 dashes in comment cause a parser error - invalid end of comment - , bug

Attributes

Attributes describe properties of elements. They provide additional metadata to help in understanding the content of the elements.

They are name-value pairs with the value in quotes.

They can appear in any order.

<company>JBJ</company> - does not work

<company>J Boyer, J</company> - works

<company><![CDATA[JBJ]]></company> - works

http://www.architier.com/courses/XML

Entities

Entities are pointers to chunks of text or other content. The parser replaces the pointer with the referenced content during parsing. Entities represent physical document structure (often files)

They are declared in the DTD and referenced in the document elements. Entity references are replaced with actual characters during parsing. An entity defines one branch of the XML tress. Markup may not span entity boundaries.

Entities always have names and may be internal (no separate physical storage unit) or external.

Internal entities make global changes easier. They are like abbreviations. You define the entity in the DTD with

```
<!ENTITY ProductName "New and improved Gizmo!">
```

Use the entity within the XML with **&ProductName;** The parser will automatically replace the string with the value declared in the DTD.

* Predefined entities include & < > ' "

External entities improve reuse of document 'components'. Breaking up a large logical document makes it easier to edit, search, download, or otherwise manipulate the content. For example, you would like to localize copyright and licensing text only once and then refer to the localized version from many different documents.

```
<!ENTITY CopyrightNotice SYSTEM
"http://wwww.architier.com/copyright.xml">
```

A reference to this entity within XML with **&CopyrightNotice;** will incorporate the entire contents of the document (and it will be parsed).

Entities may reference non-XML data in an unparsed entity with an associated notation that describes the entity format

```
<!ENTITY OurLogo SYSTEM
"http://wwww.architier.com/images/logo.gif" NDATA GIF>
<!ENTITY StopSign PUBLIC "-//ISO//GIF Stop//EN" — must be unique
"http://www.symbols.com/traffic/stop.gif" NDATA GIF>
```

NDATA allows XML system to understand that the data is "optional data" & therefore parser will not try to parse it out - you can specify what program to use to open/edit it

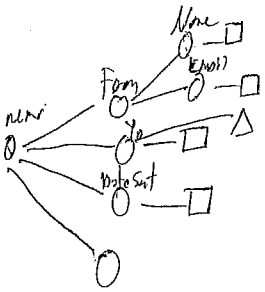
PUBLIC stipulates that following is a URN

URN = Universal Resource Name

PUBLIC - URN

SYSTEM - URL

DOCUMENT STRUCTURE WITH DTD AND SCHEMA



Description

A Document Type Definition (DTD) is a set of syntax rules for tags. It tells you

- what tags you can use in a document (vocabulary),
- what order they should appear in (sequencing),
- which tags may appear inside other ones (child hierarchy),
- which tags have attributes, and so on.

It is a formal definition of the element types, attributes, and entities allowed in a document of a specific type.

+ = 1 or more
? = 0 or 1
* = 0 or more

#IMPLIED = optional

Checking document validity with a DTD improves robustness of automatic processing of documents or data, e.g. check user input or a style sheet may rely on certain tag names and sequencing.

memo.dtd

```
<!ELEMENT memo (From, To+, DateSent,
                Subject?, Message, CC*)>
<!ELEMENT From (Name, EMail)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT EMail (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT DateSent (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ATTLIST To Priority CDATA "3" >
<!ATTLIST Message Language
                (English|French|German) "English" >
<!ATTLIST memo DigitalSignature CDATA #IMPLIED>
```

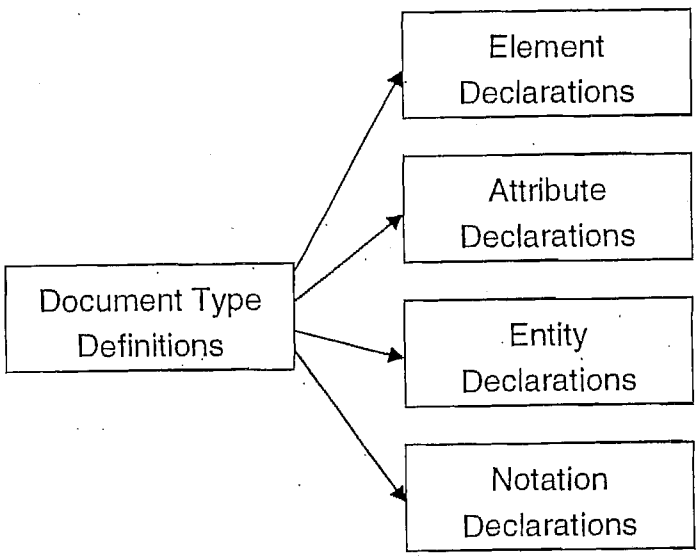
Attributes

memo.xml

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
<From>
<Name>Ike van Cruyningen</Name>
<EMail>ikevc@architier.com</EMail>
</From>
<To Priority="1">XML Class</To>
<To Priority="2">My Coordinator</To>
<DateSent>2001-11-16</DateSent>
<Message Language="English">Does it all make
sense?</Message>
</memo>
```

Rule with DTD's – Stupid Rule – It is better to Allow current tool/technology to use the standard format rather than write your own.

* Whenever possible use an existing standard *



Industry or Domain-Specific Standards

Industry-specific standards simplify communication between trading partners. Metcalfe's Law: The more people that share a protocol, the greater the value to all the participants.

Here are many publicly available schemas. For more comprehensive lists see www.schema.net and several other Web directories.

- Math ML
- Chemical ML
- Astronomy ML
- Astronomical Instrument ML
- Biosequence ML
- Newspaper Classified ML
- Weather Observation ML
- Music ML
- Genealogical ML
- Synchronized Multimedia Integration Language
- Precision Graphics Markup Language
- Web Interface Definition Language

E-Commerce

(excerpted from CNet and InfoWorld)

CommerceNet Common Business Language

Content definition: CommerceNet is working to define data elements common to a variety of commerce transactions. This so-called Commerce Core would define how to tag things like company name and address, price, item, and quantity.

Information exchange: Open, text-based XML is ideal for exchanging transaction information from one server to another. CommerceNet proposes using the XML-based Common Business Language (CBL) to describe product- and service-catalog software, metadata about business rules and systems, and software for forms and messages. Much of the CBL is drawn from existing Electronic Data Interchange (EDI) dictionaries that identify agreed-upon terms such as invoices and purchase orders. But CBL goes beyond EDI's business-to-

business focus to include retail transactions and the horizontal supply-chain--from manufacturer to wholesaler to retailer.

One such CBL application is the Product Information Exchange (PIX) specification for catalog interoperability. CommerceNet designed PIX to help manufacturers and their distributors exchange product data more easily.

CommerceXML (cXML.org) from Ariba is trying to define standards for purchase orders, invoices, catalogs, and other business documents.

BizTalk from Microsoft provides guidelines for creating standardized, interoperable XML schemas.

RosettaNet (eConcert) defines a set of Partner Interface Process specifications using DTDs for services, transactions, and messages.

Open Buying on the Internet (OBI): A standard for international business-to-business purchasing of goods through the Internet. OBI is based on current Internet standards such as HTML, SSL (for security), SET (for credit-card transactions), and X.509 (for digital certificates). Among OBI's supporters are Commerce One, Connect, Intelisys, InterWorld, Microsoft, Netscape, Open Market, and Oracle.

Open Trading Protocol (OTP): A consistent, interoperable environment for selling to consumers on the Web. Rules will range from how to offer items for sale to payment choices to product delivery, receipts, and problem resolution. OTP is backed by MasterCard International, DigiCash, CyberCash, Hewlett-Packard, IBM, AT&T Universal Card, Netscape, the Royal Bank of Canada, and a number of other financial institutions and technology companies.

Internet Content and Exchange (ICE): Vignette, Firefly Network, and a number of other companies--including Microsoft--are developing a specification called ICE to enable the site-to-site exchange of online assets, whether those are content, applications, or metadata. ICE will leverage existing standards, including OPS/P3P (for trusted exchange of personal information), CDF, OSD, XML-Data , and RDF.

Referring to DTDs in XML

DTDs are referenced before the first entity to be parsed, the document entity, with the DOCTYPE statement. They may reference externally shared DTDs

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM
"http://www.architier.com/dtd/memo.dtd">
```

or for DTDs stored in entity catalogs

```
<?xml version="1.0"?>
<!DOCTYPE memo PUBLIC "-//ARCHITIER//DTD MEMO//EN"
"http://www.architier.com/dtd/memo.dtd">
```

or internal DTDs defined for one document (avoid!)

```
<?xml version="1.0"?>
<!DOCTYPE memo [
<!ELEMENT memo (From, To+, Subject?, Message, CC*)>
<!ELEMENT From (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
]><memo>
<From>Ike van Cruyningen</From>
<To>XML Class</To>
<Subject>XML</Subject>
<Message>I hope everything is clear so far. Is your brain
getting full? Are you ready for a break?</Message>
</memo>
```

Might provide both internal and external parts and then the internal preempts the external. Entity declarations for graphics are often internal.

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "http://www.w3c.com/dtd/memo.dtd" [
<!ENTITY OurLogo SYSTEM
"http://www.architier.com/images/logo.gif" NDATA GIF>
]>
<memo>
...
</memo>
```

Elements

```
<!ELEMENT memo (From, To+, DateSent,
                Subject?, Message, CC*)>
```

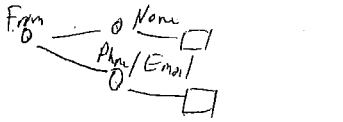
<!ELEMENT ElementTypeName Contents>

Contents may be

- nested elements (*list of allowed element type names separated by commas*)
- character data (#PCDATA),
- mixed character data and elements (#PCDATA | *list of allowed element types*),
- EMPTY for an element which takes only attributes,
- ANY for completely unstructured elements (avoid!)

Sequencing of child elements:

- , comma separates elements in a sequence
- | vertical bar separates alternatives
- () parenthesis create a content particle or group



Cardinality of child elements:

- ? question mark following a particle means it is optional (0 or 1)
- * asterisk following a particle means it is optional and repeatable (0 or more)
- + plus following a particle means it is required and repeatable (1 or more)

Attributes

```
<!ATTLIST To Priority CDATA "3" >
<!ATTLIST Message Language
                (English|French|German) "English" >
<!ATTLIST memo DigitalSignature CDATA #IMPLIED>
```

<!ATTLIST ElementTypeName AttrName AttrType AttrDefault>

Basic attribute types:

- CDATA – any character data with quote, ampersand, and other character entities properly used.
- NMTOKEN, NMTOKENS – character string limited to valid characters in names; letters, digits, period, dash, underscore, colon. Cannot be empty. no spaces
- ID – names that must be unique in the document. Used for cross-references.
- IDREF, IDREFS – a reference or link to an ID
- ENTITY, ENTITIES – reference to an entity
- Enumeration – set of alternatives like (red | white | blue)
- NOTATION – list of declared notations (GIF | JPEG | PNG)

Attribute defaults:

- #REQUIRED – must supply a value
- #IMPLIED – any appropriate value optional
- #FIXED fixedValue – one specific value only useful for version control
- defaultValue – default if no other value supplied

<!ELEMENT Chapter (Section, Figure, ...)>
<!ATTLIST Chapter ChID ID #REQUIRED>
<!ELEMENT TOCchapter (TOCSection, ...)>
<!ATTLIST TOCchapter ChIDRef IDREF #REQUIRED>

Entities

Declare a regular entity in the DTD

```
<!ENTITY ProductName "InternalLiteralEntityValue">

<!ENTITY OurLogo SYSTEM
"http://wwww.architier.com/images/logo.gif"NDATA GIF>

<!ENTITY StopSign PUBLIC "-//TrafficSymbols//Stop//EN" URN
"http://wwww.symbols.com/traffic/stop.gif" NDATA GIF> URL
```

You use a regular entity in the **XML** file with &EntityName;

Parameter entities are for use within DTDs—they cannot be accessed in an XML file, e.g.

```
<!ENTITY % WestCoastStates "(CA | OR | WA | AK | HI)">
```

Refer to a *parameter* entity in the **DTD** with %WestCoastStates;

Notation

A **notation** defines the type or data content of an external non-parsed entity. They facilitate the processing of external information that is not in XML format (e.g. binary files, image files, sound files, etc). They tell the parser what to do with information of a particular format or notation—usually by assigning an external application to process it.

```
<!NOTATION NotationName SYSTEM "ApplicationForDataType">

<!NOTATION GIF SYSTEM "paint.exe">
```

Validation

One on-line XML validator that requires Internet Explorer is available from Microsoft. Go to http://msdn.microsoft.com/xml. Look in the XML downloads section for the XML validator. This is a simple HTML file that turns on the validateOnParse flag in the built-in parser.

A highly rated portable command line and programmatic parser that does DTD validation and also supports the latest Schema standard is available from xml.apache.org..

parser.validateOnParse = true

validate.htm on their website

Element versus Attribute

Use character data for what people (or databases) expect to see in the document in every rendition.

Element can contain subtree of other elements and repeating values, i.e. structure.

DTD can enforce structure and ordering of nested elements, whereas attributes can occur in any order.

Attribute values are typically single values with no explicit structure. They cannot contain elements or 'sub-attributes'.

Attribute values can be checked by DTD.

Attributes represent properties of objects, whereas elements represent parts of objects.

Schema

A database schema is based on a rigid data model (tables with rows and columns) but has strong data typing, referential integrity constraints, and triggers or rules. A DTD is very good for defining the structure of a document, but it is quite weak in specifying data types and constraints. It also has a completely different syntax from XML. This has led to the XML-Data, Document Content Description (DCD), Schema for Object-oriented XML (SOX), and Document Definition Markup Language (DDML) proposals to support a new schema definition language. W3C is putting all this together to produce a new schema recommendation:

The XML schema language shall be:

1. more expressive than XML DTDs;
2. expressed in XML;
3. self-describing;
4. usable by a wide variety of applications that employ XML;
5. straightforwardly usable on the Internet;

6. optimized for interoperability;
7. simple enough to implement with modest design and runtime resources;
8. coordinated with relevant W3C specs (XML Information Set, Links, Namespaces, Pointers, Style and Syntax, as well as DOM, HTML, and RDF Schema).

Structural requirements

The XML schema language must define:

1. mechanisms for constraining document structure (namespaces, elements, attributes) and content (datatypes, entities, notations);
2. mechanisms to enable inheritance for element, attribute, and datatype definitions;
3. mechanism for URI reference to standard semantic understanding of a construct;
4. mechanism for embedded documentation;
5. mechanism for application-specific constraints and descriptions;
6. mechanisms for addressing the evolution of schemata;
7. mechanisms to enable integration of structural schemas with primitive data types.

Datatype requirements

The XML schema language must:

1. **provide for primitive data typing, including byte, date, integer, sequence, SQL & Java primitive data types, etc.**;
2. define a type system that is adequate for import/export from database systems (e.g., relational, object, OLAP);
3. distinguish requirements relating to lexical data representation vs. those governing an underlying information set;
4. **allow creation of user-defined datatypes**, such as datatypes that are derived from existing datatypes and which may constrain certain of its properties (e.g., range, precision, length, mask).

Conformance

The XML schema language must:

1. describe the responsibilities of conforming processors;
2. define the relationship between schemas and XML documents;
3. define the relationship between schema validity and XML validity;
4. define the relationship between schemas and XML DTDs, and their information sets;

5. define the relationship among schemas, namespaces, and validity;
6. define a useful XML schema for XML schemas;

DTD Review: memo.xml

```
<?xml version="1.0"?>
<!DOCTYPE memo SYSTEM "memo.dtd">
<memo>
<From>
<Name>Ike van Cruyningen</Name>
<EMail>ikevc@architier.com</EMail>
</From>
<To Priority="1">XML Class</To>
<To Priority="2">My Coordinator</To>
<DateSent>2000-11-16</DateSent>
<Message Language="English">Does it all make
sense?</Message>
</memo>
```

memo.dtd

```
<!ELEMENT memo (From, To+, DateSent,
                Subject?, Message, CC*)>
<!ELEMENT From (Name, EMail)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT EMail (#PCDATA)>
<!ELEMENT To (#PCDATA)>
<!ELEMENT DateSent (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Message (#PCDATA)>
<!ELEMENT CC (#PCDATA)>
<!ATTLIST To Priority CDATA "3" >
<!ATTLIST Message Language
                (English|French|German) "English" >
<!ATTLIST memo DigitalSignature CDATA #IMPLIED>
```

XML-Data Example

Microsoft developed an early version of schema called XML-Data. This is the basis for the W3C standard, but does not include all the features in the standard. It is built into IE5, and so has wide distribution.

MemoXMLData.xml

```
<?xml version="1.0"?>
<memo xmlns="x-schema:MemoXMLData.xdr">
<From>
<Name>Ike van Cruyningen</Name>
<EMail>ikevc@architier.com</EMail>
</From>
<To Priority="1">XML Class</To>
<To Priority="2">My Coordinator</To>
<DateSent>2001-11-16</DateSent>
<Message Language="English">Does it all make
sense?</Message>
</memo>
```

MemoXMLData.xdr

```
<?xml version="1.0"?>
<Schema name="MemoSchema"
   xmlns="urn:schemas-microsoft-com:xml-data"
   xmlns:dt="urn:schemas-microsoft-com:datatypes">

 <ElementType name="Name" content="textOnly"
              model="closed" dt:type="string" />

 <ElementType name="EMail" content="textOnly"
              model="closed" dt:type="string" />

 <ElementType name="From" content="eltOnly" model="closed">
     <element type="Name" />
     <element type="EMail" />
 </ElementType>

 <AttributeType name="Priority" dt:type="int" />

 <ElementType name="To" content="textOnly"
              model="closed" dt:type="string" >
     <attribute type="Priority" />
```

```
</ElementType>

<ElementType name="DateSent" content="textOnly"
             model="closed" dt:type="date" />

<ElementType name="Subject" content="textOnly"
             model="closed" dt:type="string" />

<AttributeType name="Language" dt:type="enumeration"
    dt:values="English French German" default="English" />

<ElementType name="Message" content="textOnly"
             model="closed" dt:type="string" >
    <attribute type="Language" />
</ElementType>

<ElementType name="CC" content="textOnly"
             model="closed" dt:type="string" />


<AttributeType name="DigitalSignature" dt:type="string"/>

<ElementType name="memo" content="eltOnly" model="closed">
    <element type="From" />
    <element type="To" maxOccurs="*" />
    <element type="DateSent" />
    <element type="Subject" minOccurs="0" maxOccurs="1" />
    <element type="Message" />
    <element type="CC" minOccurs="0" maxOccurs="*" />
    <attribute type="DigitalSignature" />
</ElementType>
</Schema>
```

XML Schema

XML Schema from W3C (final standard May 2, 2001) is based on XML-Data, but is much more general. It includes a complete type system (with derivation), more flexibility in data type facets, a regular expression language for matching input, groups and unions of types, etc.

Rather than using a DOCTYPE statement, you refer to a schema file with attributes on the root element of your XML:

MemoSchema.xml

```
<?xml version="1.0"?>
<memo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="memoSchema.xsd">
<From>
<Name>Ike van Cruyningen</Name>
</From>
<To Priority="1">XML Class</To>
<To Priority="2">My Coordinator</To>
<DateSent>2001-07-30</DateSent>
<Message Language="English">
Does it all make sense?</Message>
</memo>
```

MemoSchema.xsd

The Schema file is an XML file with a number of type declarations and one instance declaration. By convention, schema files have the suffix 'xsd'.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

<xsd:element name="memo" type="MemoType"/>

<xsd:complexType name="FromType">
  <xsd:sequence>
  <xsd:element name="Name" type="xsd:string" />
  <xsd:element name="EMail" type="xsd:string"
    minOccurs="0"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="MemoType">
```

```
<xsd:sequence>
<xsd:element name="From" type="FromType" />

<xsd:element name="To" maxOccurs="unbounded">
   <xsd:complexType>
    <xsd:simpleContent>
     <xsd:extension base="xsd:string">
      <xsd:attribute name="Priority" type="xsd:int" />
     </xsd:extension>
    </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>

<xsd:element name="DateSent" type="xsd:date"/>
<xsd:element name="Subject" type="xsd:string"
    minOccurs="0"/>

<xsd:element name="Message" >
   <xsd:complexType>
    <xsd:simpleContent>
     <xsd:extension base="xsd:string">
      <xsd:attribute name="Language" type="LanguageType"
         use="optional" default="English" />
     </xsd:extension>
    </xsd:simpleContent>
   </xsd:complexType>
</xsd:element>

<xsd:element name="CC" type="xsd:string" minOccurs="0"
    maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="DigitalSignature"
    type="xsd:string"/>
</xsd:complexType>

<xsd:simpleType name="LanguageType">
 <xsd:restriction base="xsd:string">
 <xsd:enumeration value="English"/>
 <xsd:enumeration value="French Canadian"/>
 <xsd:enumeration value="German"/>
 </xsd:restriction>
</xsd:simpleType>
```

```
</xsd:schema>
```

Built-in Types

Simple Type	Examples (delimited by commas)	Notes
string	Sample string	
normalizedString	Sample string	Newline, tab and carriage-return characters in a normalizedString type are converted to space characters before schema processing.
token	Sample string	As normalizedString, and adjacent space characters are collapsed to a single space character, and leading and trailing spaces are removed.
byte	-1, 126	see (2)
unsignedByte	0, 126	see (2)
base64Binary	GpM7	
hexBinary	0FB7	
integer	-126789, -1, 0, 1, 126789	see (2)
positiveInteger	1, 126789	see (2)
negativeInteger	-126789, -1	see (2)
nonNegativeInteger	0, 1, 126789	see (2)
nonPositiveInteger	-126789, -1, 0	see (2)
int	-1, 126789675	see (2)
unsignedInt	0, 1267896754	see (2)
long	-1, 12678967543233	see (2)
unsignedLong	0, 12678967543233	see (2)
short	-1, 12678	see (2)
unsignedShort	0, 12678	see (2)

decimal	-1.23, 0, 123.4, 1000.00	see (2)
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	equivalent to single-precision 32-bit floating point, NaN is "not a number", see (2)
double	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN	equivalent to double-precision 64-bit floating point, see (2)
boolean	true, false 1, 0	
time	13:20:00.000, 13:20:00.000-05:00	see (2)
dateTime	1999-05-31T13:20:00.000-05:00	May 31st 1999 at 1.20pm Eastern Standard Time which is 5 hours behind Co-Ordinated Universal Time, see (2)
duration	P1Y2M3DT10H30M12.3S	1 year, 2 months, 3 days, 10 hours, 30 minutes, and 12.3 seconds
date	1999-05-31	see (2)
gMonth	--05--	May, see (2) (5)
gYear	1999	1999, see (2) (5)
gYearMonth	1999-02	the month of February 1999, regardless of the number of days, see (2) (5)
gDay	---31	the 31st day, see (2) (5)
gMonthDay	--05-31	every May 31st, see (2) (5)
Name	shipTo	XML 1.0 Name type
QName	po:USAddress	XML Namespace QName
NCName	USAddress	XML Namespace NCName, i.e. a QName without the prefix and colon

anyURI	http://www.example.com/, http://www.example.com/doc.html#ID5	
language	en-GB, en-US, fr	valid values for xml:lang as defined in XML 1.0
ID		XML 1.0 ID attribute type, see (1)
IDREF		XML 1.0 IDREF attribute type, see (1)
IDREFS		XML 1.0 IDREFS attribute type, see (1)
ENTITY		XML 1.0 ENTITY attribute type, see (1)
ENTITIES		XML 1.0 ENTITIES attribute type, see (1)
NOTATION		XML 1.0 NOTATION attribute type, see (1)
NMTOKEN	US, Brésil	XML 1.0 NMTOKEN attribute type, see (1)
NMTOKENS	US UK, Brésil Canada Mexique	a whitespace separated list of NMTOKEN's, see (1)

Notes: (1) To retain compatibility between XML Schema and XML 1.0 DTDs, the simple types ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATION, NMTOKEN, NMTOKENS should only be used in attributes. (2) A value of this type can be represented by more than one lexical format, e.g. 100 and 1.0E2 are both valid float formats representing "one hundred". However, rules have been established for this type that define a canonical lexical format, see XML Schema Part 2. (5) The "g" prefix signals time periods in the Gregorian calender.

Simple Types

Simple types are modifications of the built-in types. You modify a built-in type by setting values of 'facets' such as

- length, minLength, maxLength for the string types
- maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits for the ordered types
- pattern, enumeration, whiteSpace for all types

Some examples of simple types include range limitations, enumerations, and patterns:

```
<xsd:simpleType name="fastCheckoutLineType">
  <xsd:restriction base="xsd:integer">
    <xsd:minInclusive value="1"/>
    <xsd:maxInclusive value="9"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="StateCodeType">
  <xsd:restriction base="xsd:string">
    <xsd:length value="2"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="LanguageType">
  <xsd:restriction base="xsd:string">
  <xsd:enumeration value="English"/>
  <xsd:enumeration value="French Canadian"/>
  <xsd:enumeration value="German"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<xsd:simpleType name="PhoneType">
   <xsd:restriction base="xsd:string">
   <xsd:pattern value="\d{3}-\d{3}-\d{4}" />
   </xsd:restriction>
</xsd:simpleType>
```

Simple types also support lists of values (c.f. IDREFS or NMTOKENS) and unions of types.

w3c.org → schema - intro to schema will provide reference on regular expressions

Complex Types

Complex types define elements with child elements or attributes (aggregates):

```
<xsd:complexType name="FromType">
  <xsd:sequence>
  <xsd:element name="Name" type="xsd:string" />
  <xsd:element name="EMail" type="xsd:string"
    minOccurs="0" maxOccurs="3" />
  </xsd:sequence>
</xsd:complexType>
```

The child elements can be enclosed in

<xsd:sequence> if elements have to appear in order (DTD comma),

<xsd:choice> where of the alternatives must appear (DTD |),

<xsd:group> which combines elements into a unit (DTD ()),

<xsd:all> where all the elements in the group may appear once or not at all, and they may appear in any order

The cardinality and value requirements are expressed with the attributes minOccurs, maxOccurs, default, fixed, and use (required, optional, prohibited)

You create complex types from other types either explicitly by naming them or anonymously by extending or restricting them. The previous example defined the complex type FromType and you would use it within another type as:

```
<xsd:complexType name="MemoType">
  <xsd:sequence>
  <xsd:element name="From" type="FromType" /> ...
```

If you won't be reusing the type definition in different contexts, you might just define it anonymously within MemoType:

```
<xsd:complexType name="MemoType">
  <xsd:sequence>
  <xsd:element name="From" >
    <xsd:complexType>
    <xsd:sequence>
    <xsd:element name="Name" type="xsd:string" />
    <xsd:element name="EMail" type="xsd:string"
```

```
      minOccurs="0" maxOccurs="3" />
    </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
  ...
```

You might also define a restriction on a simple type anonymously, or extend a simple type with an attribute:

```
<xsd:element name="internationalPrice">
  <xsd:complexType>
   <xsd:simpleContent>
    <xsd:extension base="xsd:decimal">
     <xsd:attribute name="currency" type="xsd:string"/>
    </xsd:extension>
   </xsd:simpleContent>
  </xsd:complexType>
</xsd:element>
```

Schema File

In your Schema file you will also define at least one instance element:

```
<xsd:element name="memo" type="MemoType"/>
```

If you have a large schema or wish to reuse portions of it, then you can break it up in several files and include, redefine, or import the derivative files:

```
<include
schemaLocation="http://www.ex.com/schemas/address.xsd"/>
<redefine
  schemaLocation="http://www.ex.com/schemas/address.xsd">
 ... changes to type definitions
</redefine>

<import namespace="http://www.eec.org/metric"
 schemaLocation="http://www.eec.org/standards/metric.xsd"
/>
```

Analogous to the DTD ID and IDREF concepts, you can define attribute and element fields to be unique, key, or keyref values within a certain scope. You use

the XPath standard (which we will discuss later in the course) to define the fields.

```
<key name="partNumKey">
   <selector xpath="r:parts/r:part"/>
   <field xpath="@partNumber"/>
</key>
```

Namespaces

Required to distinguish conflicting names from independently developed document components and Schema. Documents may contain elements or attributes with the same names, but different semantics.

Need a mechanism to enhance names so they are universally unique. Rely on central URI authority to provide global uniqueness.

You declare namespaces on the root element or a branch of the XML tree with xmlns:prefix="URI". Use the namespace in the elements by qualifying the element name with the prefix. The default namespace does not have a prefix.

```
<?xml version="1.0" ?>
<order xmlns="http://www.ecommerce.org/PurchaseOrder"
xmlns:metric="http://www.eec.org/metric"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ecommerce.org/PurchaseOrder
PO.xsd">
<SKU>123456789</SKU><Qty>5</Qty>
<metric:units>kilogram</metric:units>
<price>12.99</price>
</order>
```

If you were to try to validate the above order, the Schema document would import the metric Schema using the following syntax:

```
<schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
  xmlns=http://www.ecommerce.org/PurchaseOrder
  targetNamespace="http://www.ecommerce.org/PurchaseOrder">

<xsd:import namespace="http://www.eec.org/metric"
 schemaLocation="http://www.eec.org/standards/metric.xsd"
/>
```

A namespace may apply only to a certain branch of the element tree, rather than the whole document. Add the xmlns attribute to an element to make it the default namespace for that branch

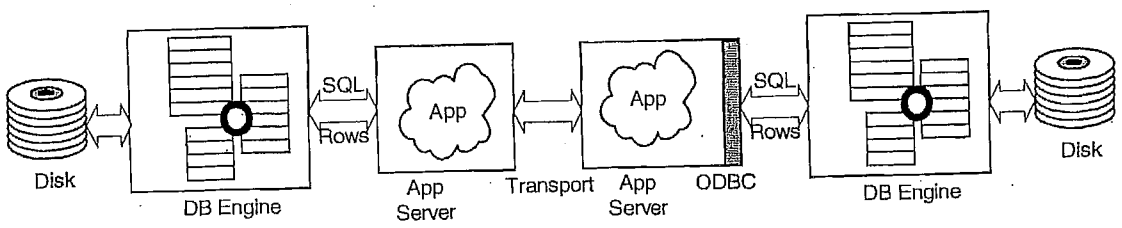
```
<OrderItem xmlns="www.architier.com/ns/">
```

or define a prefix for use in the element and all nested elements (this prefix overrides any previous definitions)

```
<edi:units xmlns:edi="www.architier.com/ns/edi">
```

COMMUNICATING INFORMATION

XML is a portable format for communicating information between disparate systems, both within an organization and between organizations. This is its greatest value to e-commerce applications. Most of this data originates in a database and has to end up in a database. This section illustrates code for implementing these types of applications.



Example

Suppose we would like to provide the capability to find nearby hospitals, retail outlets, distribution locations, or yellow pages information. You would like to support both on-line people-oriented forms and machine queries.

ZipCode	Latitude	Longitude
94022	37.381432	122.125754
94024	37.354745	122.086205
94028	37.378859	122.208131
94306	37.418009	122.127375

Name	City	State	ZipCode
El Camino Healthcare	Mountain View	CA	94040
Sequoia Hospital	Redwood City	CA	94062
Kaiser Foundation	Redwood City	CA	94063
Stanford University Hospital	Palo Alto	CA	94305

Generating XML on the Fly

Communication between the systems often uses HTTP over TCP/IP since it is so widely available. A piece of information could be requested with a standard HTTP GET or POST using an HTML form something like:

```
<html>
<head>
<title>Facility Locator</title>
</head>
<body>
<img src="images/ArchitierLarge.gif">
<h3>Facility Locator</h3>
<p>Please enter your zip code to find the locations near
you: </p>
<form action="LookupFacilities.asp" method="Get">
<input type="text" size="5" maxlength="5" name="zip">
<input type="submit" value="Find Closest HTML">
</form>
<hr />

<form action="LookupFacilitiesXML.asp" method="Get">
<input type="text" size="5" maxlength="5" name="zip">
<input type="submit" value="Find Closest XML">
</form>
<hr />

<form
action="http://127.0.0.1:8080/servlet/COM.architier.Example
s.LookupFacility.LookupFacilityServlet" method="Get">
<input type="text" size="5" maxlength="5" name="zip">
<input type="submit" value="Servlet XML">
</form>
<hr />

</body>
</html>
```

Any tool or technology you use to create HTML can be used to create XML. The most popular technologies are server-side scripting (JSP, servlets, and ASP) and template approaches (ColdFusion and w3-mSQL).

Several server vendors are developing specific XML servers that make it easy to map specific database fields to XML fields.

Active Server Pages

LookupFacilities.asp

```
<html>
<head>
<title>Nearby Facilities</title>
</head>
<body>
<%
If Len(request.Item("zip")) = 5 Then
   ' If the client entered a Zip code, look up the
   ' corresponding lat and long in the ZIP database table
   set connection = server.CreateObject("ADODB.Connection")
   connection.Open
      "DSN=LocateFacility;UID=Reader;PWD=GetLoc"
   set rsLatLong = server.CreateObject("ADODB.Recordset")
   rsLatLong.Open _
      "SELECT  Latitude, Longitude " & _
      "FROM Zip " & _
      "WHERE ZipCode = " & request.Item("zip"), connection
   If Not rsLatLong.EOF Then
      ' Now find the closest 25 facilities,
      ' listed in order of ~distance
      set rsLoc = server.CreateObject("ADODB.Recordset")
      rsLoc.MaxRecords = 25
      rsLoc.Open _
         "SELECT  f.Name, f.City, f.State, Distance = " & _
         "      (SQRT(POWER(z.Latitude - " &
         rsLatLong("Latitude") & ", 2) + " & _
         "      POWER(z.Longitude - " &
         rsLatLong("Longitude") & ", 2)) * 69.04124) " & _
         "FROM Zip z, Facility f " & _
         "WHERE    z.ZipCode = f.ZipCode " & _
         "ORDER BY Distance", connection
      If Not rsLoc.EOF Then
         ' Create an HTML table containing the
         ' results of the query
         %>
         The facilities closest to you are
         <table border=0 cellpadding=5>
         <tr><th>Name</th><th>City</th>
            <th>State</th><th>Distance</th></tr>
```

```
        <% Do While not rsLoc.EOF %>
           <tr><td> <% = rsLoc("Name") %> </td>
           <td> <% = rsLoc("City") %> </td>
           <td> <% = rsLoc("State") %> </td>
           <td> (~<% = Left(rsLoc("Distance"),4) %> miles)
           </td></tr>
        <% rsLoc.MoveNext
        Loop
        %> </table> <%
     Else
        response.Write("Sorry we have no locations " & _
           "near ZIP code " & request.Item("zip"))
     End If
     rsLoc.Close
  Else
     response.Write("Sorry we have no data for " & _
        "ZIP code " & request.Item("zip"))
  End If
  rsLatLong.Close
  connection.Close
Else
  response.Write("Please enter a five digit ZIP code.")
End If %>
</body>
</html>
```

LookupFacilitiesXML.asp

```
<?xml version="1.0"?>
<facilities>
<%
If Len(request.Item("zip")) = 5 Then
   ' If the client entered a Zip code, look up the
   ' corresponding lat and long in the ZIP database table
   set connection = server.CreateObject("ADODB.Connection")
   connection.Open
      "DSN=LocateFacility;UID=Reader;PWD=GetLoc"
   set rsLatLong = server.CreateObject("ADODB.Recordset")
   rsLatLong.Open _
      "SELECT  Latitude, Longitude " & _
      "FROM Zip " & _
      "WHERE ZipCode = " & request.Item("zip"), connection
   If Not rsLatLong.EOF Then
```

```
    ' Now find the closest 25 facilities,
    ' listed in order of ~distance
    set rsLoc = server.CreateObject("ADODB.Recordset")
    rsLoc.MaxRecords = 25
    rsLoc.Open _
       "SELECT  f.Name, f.City, f.State, Distance = " & _
       "     (SQRT(POWER(z.Latitude - " & _
       rsLatLong("Latitude") & ", 2) + " & _
       "     POWER(z.Longitude - " & _
       rsLatLong("Longitude") & ", 2)) * 69.04124) " & _
       "FROM Zip z, Facility f " & _
       "WHERE   z.ZipCode = f.ZipCode " & _
       "ORDER BY Distance", connection
    If Not rsLoc.EOF Then
       ' For each facility create a data record
       " with name, city, state, and distance
       Do While not rsLoc.EOF %>
          <facility>
          <name><% = rsLoc("Name") %></name>
          <city> <% = rsLoc("City") %> </city>
          <state> <% = rsLoc("State") %> </state>
          <distance><% = Left(rsLoc("Distance"),4) %>
             </distance>
          </facility>
       <% rsLoc.MoveNext
       Loop
    Else
       response.Write("<status code=""NoLocations"">" & _
         "Sorry we have no locations near ZIP code " & _
            request.Item("zip") & "</status>")
    End If
    rsLoc.Close
  Else
    response.Write("<status code=""NoData"">Sorry we" & _
      "have no data for ZIP code " & _
      request.Item("zip") & "</status>")
  End If
  rsLatLong.Close
  connection.Close
Else
  response.Write("<status code=""InvalidRequest"">" _
    "Please enter a five digit ZIP code.</status>")
End If %>
```

```
</facilities>
```

Servlets

LookupFacilityServlet.java

```
package COM.architier.XMLSamples.LookupFacility;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class LookupFacilityServlet extends HttpServlet {

public String getServletInfo()
{
   return "Example servlet to look up nearby facilities";
}

public void doGet(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
    doGetOrPost(req,resp);
}
public void doPost(HttpServletRequest req,
HttpServletResponse resp)
throws ServletException, IOException {
   doGetOrPost(req,resp);
}
private void doGetOrPost(HttpServletRequest req,
                  HttpServletResponse res)
            throws ServletException, IOException {
res.setContentType("text/xml");
ServletOutputStream out = res.getOutputStream();
out.println("<?xml version=\"1.0\"?><facilities>");
try {
    // Load the JDBC driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // Connect to the database and execute statement
    Connection con = DriverManager.getConnection(
         "jdbc:odbc:LocateFacility","Reader","GetLoc");
    Statement sqlcmd = con.createStatement();
```

```
    sqlcmd.setMaxRows(25);
    ResultSet r = sqlcmd.executeQuery(
    "SELECT Latitude, Longitude FROM Zip WHERE ZipCode = '"
       + req.getParameter("zip") + "'");
   if (!r.next()) {
      out.println("<status code=\"NoData\">Sorry we
            have no data for ZIP code " +
         req.getParameter("zip") + "</status>");
   } else {
      ResultSet rl = sqlcmd.executeQuery(
"SELECT  f.Name, f.City, f.State, Distance = " +
"(SQRT(POWER(z.Latitude - " + r.getString(1) + ", 2) + " +
" POWER(z.Longitude - " + r.getString(2) + ", 2))*69.0) " +
       "FROM   Zip z, Facility f " +
       "WHERE  z.ZipCode = f.ZipCode ORDER BY Distance");
       if (!rl.next()) {
          out.println("<status code=\"NoLocations\">
             Sorry we have no locations near ZIP code " +
             req.getParameter("zip") + "</status>");
       } else {
          do {
             out.println(
             "<facility>" +
                "<name>" + rl.getString(1) + "</name>" +
                "<city>" + rl.getString(2) + "</city>" +
                "<state>" + rl.getString(3) + "</state>" +
                "<distance>" + rl.getString(4) +
                  "</distance>" +
               </facility>");
             } while (rl.next());
       }
      rl.close();
    }
    out.println("</facilities>");
    r.close();
    sqlcmd.close();
    con.close();
} catch (java.lang.Exception ex) {
   System.err.println("Exception: " + ex.getMessage());
   out.println("<status code=\"Exception\">" +
      ex.getMessage() + "</status>");
   out.println("</facilities>");
}
```

```
}
}
```

Java Server pages

Java Server Pages simplify servlets by providing syntax almost identical to ASP and compiling into servlets at runtime.

Alternatives are template-based solutions both for specifying the queries and parameters to be sent to the database server, and for the XML format to be produced.

```
<?xml version="1.0"?>
<facilities>
<%@ page import="
    java.io.*,
    javax.servlet.*,
    javax.servlet.http.*,
    java.sql.*"
%>
<%
try {
    // Load the JDBC driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // Connect to the database and execute statement
    Connection con = DriverManager.getConnection(
       "jdbc:odbc:LocateFacility","Reader","GetLoc");
    Statement sqlcmd = con.createStatement();
    sqlcmd.setMaxRows(25);
    ResultSet r = sqlcmd.executeQuery(
    "SELECT Latitude, Longitude FROM Zip WHERE ZipCode = '"
       + request.getParameter("zip") + "'");
   if (!r.next()) {
      out.println("<status code=\"NoData\">Sorry we
            have no data for ZIP code " +
         req.getParameter("zip") + "</status>");
   } else {
      ResultSet rl = sqlcmd.executeQuery(
      "SELECT  f.Name, f.City, f.State, Distance =
      (SQRT(POWER(z.Latitude - " + r.getString(1) +
      ", 2) + POWER(z.Longitude - " + r.getString(2) +
```

```
    ", 2))*69.0) FROM Zip z, Facility f
    WHERE z.ZipCode = f.ZipCode ORDER BY Distance");

     if (!rl.next()){
        out.println("<status code=\"NoLocations\">
           Sorry we have no locations near ZIP code " +
           req.getParameter("zip") + "</status>");
     } else {
          do {
           out.println(
            "<facility>" +
               "<name>" + rl.getString(1) + "</name>" +
               "<city>" + rl.getString(2) + "</city>" +
               "<state>" + rl.getString(3) + "</state>" +
               "<distance>" + rl.getString(4) +
                 "</distance>" +
              </facility>");
            } while (rl.next());
      }
     rl.close();
   }
   r.close();
   sqlcmd.close();
   con.close();
}
catch (java.lang.Exception ex)
{
   System.err.println("Exception: " + ex.getMessage());
   out.println("<status code=\"Exception\">" +
      ex.getMessage() + "</status>");
}
%>
</facilities>
```

Similar to tables in HTML

Parsing XML

At the other end of the connection the XML has to be parsed and then an action has to be taken. An XML processor or parser can either

1. Parse the entire element tree creating an object tree that is manipulated by the program (DOM), or

2. Send an event to the code on each significant parse event, such as start element, end element, etc. (SAX)

The DOM approach is better for structural document modifications (sorting), or when multiple passes over the document elements are required (pagination and cross-reference updates).

+ Flexible with multi-pass read & write operations
+ Easy programming model
- Heavy memory usage
- Have to read all XML before processing starts.

The SAX approach is much more efficient, particularly for very large documents, when performing element by element tasks like counting or searching.

+ Very fast character scan (no parsing)
+ Tiny memory footprint (4Kbyte scanner and 100 byte buffer)
- Single pass, read-only
- Unusual event-driven programming model
- Maintaining state is usually done with global variables.

DOM

The Document Object Model is an application-programming interface (API) to work with a representation of a structured document. You use it in programs to examine of manipulate document elements, attributes, and text. It defines

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects (behavior and attributes)
- the relationships and collaboration among the interfaces and objects.

The interfaces are defined in the Corba Interface Definition Language (IDL) and they are implemented in Java, JavaScript, COM, and other forms. The document is represented as a tree with nodes for elements, attributes, text, etc. You get properties of a node in the tree with

```
R/O,R/W  Prop/Method  Type      Name
readonly attribute    DOMString nodeName;
         attribute    DOMString nodeValue;
readonly attribute    unsigned short nodeType;
```

The values of `nodeName`, `nodeValue`, and `attributes` vary according to the node type as follows:

Node type	nodeName	nodeValue	attributes
Element	tagName	null	NamedNodeMap
Attr	name of attribute	value of attribute	null
Text	#text	content of the text node	null
CDATASection	#cdata-section	content of the CDATA Section	null
EntityReference	name of entity referenced	null	null
Entity	entity name	null	null
ProcessingInstruction	target	entire content excluding the target	null
Comment	#comment	content of the comment	null
Document	#document	null	null
DocumentType	document type name	null	null
DocumentFragment	#document-fragment	null	null
Notation	notation name	null	null

Navigation through the tree is either with a search access

```
ReturnType  NodeType.MethodName(arguments)
NodeList anyElement.getElementsByTagName("TagName");
Node  anyNamedNodeMap.getNamedItem("NodeName");
```

or with positional access

```
R/O,R/W  Prop/Method  Type                Name
readonly attribute  Node                  firstChild;
readonly attribute  Node                  lastChild;
readonly attribute  Node                  previousSibling;
readonly attribute  Node                  nextSibling;
readonly attribute  NodeList              childNodes;
readonly attribute  NamedNodeMap          attributes;
readonly attribute  Document              ownerDocument;
```

The utility interfaces are:

```
interface NodeList {
  Node                item(in unsigned long index);
  readonly attribute unsigned long    length;
};

interface NamedNodeMap {
  Node                getNamedItem(in DOMString name);
  Node                setNamedItem(in Node arg)
  Node                removeNamedItem(in DOMString name)
  Node                item(in unsigned long index);
  readonly attribute unsigned long    length;
};
```

documentElement.getElementsByTagName("ZIP").item(0).FirstChild.nodeValue

[Positional]
documentElement.firstChild.childNodes.item(1).firstChild.lastChild.firstChild.nodeValue

documentElement.getElementsByTagName([redacted])("Bill")

Interface for Generic Node

```
interface Node {
  readonly attribute DOMString        nodeName;
           attribute DOMString        nodeValue;
  readonly attribute unsigned short   nodeType;
  readonly attribute Node             parentNode;
  readonly attribute NodeList         childNodes;
  readonly attribute Node             firstChild;
  readonly attribute Node             lastChild;
  readonly attribute Node             previousSibling;
  readonly attribute Node             nextSibling;
  readonly attribute NamedNodeMap     attributes;
  readonly attribute Document         ownerDocument;
  Node               insertBefore(in Node newChild,
                                  in Node refChild)
  Node               replaceChild(in Node newChild,
                                  in Node oldChild)
  Node               removeChild(in Node oldChild)
  Node               appendChild(in Node newChild)
  boolean            hasChildNodes();
  Node               cloneNode(in boolean deep)
};
```

deep set to false only clones the current node
" " " true clones entire branch

Interface for any Element

```
interface Element : Node {
  readonly attribute DOMString        tagName;
  DOMString          getAttribute(in DOMString name);
  void               setAttribute(in DOMString name,
                                  in DOMString value)
  void               removeAttribute(in DOMString name)
  Attr               getAttributeNode(in DOMString name);
  Attr               setAttributeNode(in Attr newAttr)
  Attr               removeAttributeNode(in Attr oldAttr)
  NodeList           getElementsByTagName(
                        in DOMString name);
  void               normalize();
};
```

Interface for Attr Node

```
interface Attr : Node {
  readonly attribute DOMString        name;
  readonly attribute boolean          specified;
           attribute DOMString        value;
};
```

Interfaces for CharacterData and Text Nodes

```
interface CharacterData : Node {
           attribute DOMString        data;
  readonly attribute unsigned long    length;
  DOMString          substringData(in unsigned long offset,
                                   in unsigned long count)
  void               appendData(in DOMString arg)
  void               insertData(in unsigned long offset,
                                in DOMString arg)
  void               deleteData(in unsigned long offset,
                                in unsigned long count)
  void               replaceData(in unsigned long offset,
                                 in unsigned long count,
                                 in DOMString arg)
};

interface Text : CharacterData {
  Text               splitText(in unsigned long offset)
};
```

Interface for Document Node

```
interface Document : Node {
  readonly attribute DocumentType      doctype;
  readonly attribute DOMImplementation  implementation;
  readonly attribute Element           documentElement;
  Element              createElement(in DOMString tagName)
  DocumentFragment     createDocumentFragment();
  Text                 createTextNode(in DOMString data);
  Comment              createComment(in DOMString data);
  CDATASection         createCDATASection(in DOMString data)
  ProcessingInstruction createProcessingInstruction(
                           in DOMString target,
                           in DOMString data)
  Attr                 createAttribute(in DOMString name)
  EntityReference      createEntityReference(
                           in DOMString name)
  NodeList             getElementsByTagName(
                           in DOMString tagname);
};
```

Here is an example to parse XML and save values to the database. It uses XML documents of the form:

```
<?xml version="1.0"?>
<facility>
<name>Palo Alto Medical Center</name>
<city>Palo Alto</city>
<state>CA</state>
<zip>94305</zip>
</facility>
```

SaveFacility.asp

```
<?xml version="1.0"?>
<FacilitySave>
<%
If Len(request.Item("XMLAddress")) > 30 Then
   set parser = Server.CreateObject("Microsoft.XMLDOM")
   parser.async = false
   parser.validateOnParse = true
   xmlstr = request.Item("XMLAddress")
   If (parser.loadXML(xmlstr)) Then
      set nameNode = parser.documentElement.
                     getElementsByTagname("name").item(0)
      set cityNode = nameNode.nextSibling
      set zipNode =  parser.documentElement.
                     getElementsByTagname("zip").item(0)
      set stateNode = zipNode.previousSibling
      If Not nameNode Is Nothing And _
         Not cityNode Is Nothing And _
         Not stateNode Is Nothing And _
         Not zipNode Is Nothing Then
         If Len(nameNode.text) > 0 And _
            Len(cityNode.text) > 0 And _
           Len(stateNode.text) > 0 And _
           Len(zipNode.text) > 0 Then
           set connection =
               server.CreateObject("ADODB.Connection")
           connection.Open
               "DSN=SaveFacility;UID=Writer;PWD=SendItUp"
           set rs = server.CreateObject("ADODB.Recordset")
           sqlStmt = "INSERT INTO dbo.Facility VALUES (" &
            zipNode.text & _
            "','" & stateNode.text & _
```

```
            "',' " & nameNode.text & _
            "',' " & cityNode.text & "')"
           ' Response.Write(sqlStmt)
           rs.Open sqlStmt, connection
           %> <status code="Success">Facility '
            <% = nameNode.text %>' in
            '<% = cityNode.text %>',
            '<% = stateNode.text %>',
            '<% = zipNode.text %>'
            successfully added.</status> <%
           set rs = Nothing
           set connection = Nothing
         Else %>
            <status code="InvalidValue">Please enter
               values for each field.
               name= <% = nameNode.nodeName %> :
               <% = nameNode.text %>
               city= <% = cityNode.text %>
               state= <% = stateNode.text %>
               zip= <% = zipNode.text %>
            </status>
         <% End If
      Else %>
         <status code="InvalidXML">Please create a node
            for name, city, state, and zip.</status>
      <% End If
   Else %>
      <status code="InvalidRequest"> Problems loading XML
document:<% = parser.parseError.reason %></status>
<%  End If
Else %>
   <status code="InvalidRequest">Please enter an address
      in XML format.</status>
<% End If %>
</FacilitySave>
```

Here is an example to print the purchase order total using a Java program with the Xerces parser from xml.apache.com:

```
import util.Arguments;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.apache.xerces.parsers.DOMParser;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.SAXNotRecognizedException;
import org.xml.sax.SAXNotSupportedException;

public class ParseXMLDOM implements ErrorHandler {

    private static boolean setValidation    = true; //defaults
    private static boolean setNameSpaces    = true;
    private static boolean setSchemaSupport = true;


    public static void main(String argv[]) {

    String uri = new String(parseCommandLine(argv));

        try {
            DOMParser parser = new DOMParser();

            parser.setFeature( "http://xml.org/sax/features/namespaces", setNameSpaces );
            parser.setFeature( "http://apache.org/xml/features/validation/schema",
                               setSchemaSupport );
            parser.setFeature( "http://xml.org/sax/features/validation", setValidation );

         ParseXMLDOM parseXML = new ParseXMLDOM();   // for Error Handler
         parser.setErrorHandler(parseXML);

         long startParse = new java.util.Date().getTime();
         parser.parse(uri);
         System.out.println("Successful DOM parse for: " + uri + " in " +
                  (new java.util.Date().getTime() - startParse) + " ms.");

         Document doc = parser.getDocument();

         /* Sample code works only for PO.xml */
         System.out.println("PO Total: " +
               doc.getElementsByTagName("Total").item(0).getFirstChild().getNodeValue());

        } catch (org.xml.sax.SAXParseException spe) {
                    System.out.println("Error: " + spe.getMessage() +
                        " at line: " + spe.getLineNumber() + " column: " +
                    spe.getColumnNumber());
        } catch (org.xml.sax.SAXNotRecognizedException ex ){
            System.out.println("Error: " + ex.getMessage());
```

```
        } catch (org.xml.sax.SAXNotSupportedException ex ){
            System.out.println("Error: " + ex.getMessage());
        } catch (org.xml.sax.SAXException se) {
            System.out.println("Error: " + se.getMessage());
        } catch (Exception e) {
            System.out.println("Error: " + e.getMessage());
        }

    } // main(String[])

    public static String parseCommandLine(String argv[]) {
        Arguments argopt = new Arguments();
        argopt.setUsage( new String[] {
                            "usage: java ParseXMLDOM (options) uri ",
                            "",
                            "options:",
                            "  -v | -V  Turn off validation [default=on]",
                            "  -s | -S  Turn off Schema support [default=on]",
                            "  -n | -N  Turn off namespace [default=on]",
                            "  -h       This help screen."} );
        if (argv.length == 0) {
            argopt.printUsage();
            System.exit(1);
        }
        argopt.parseArgumentTokens(argv , new char[] { 'p'} );

        int   c;
        String arg = argopt.getlistFiles();
        while ( (c =  argopt.getArguments()) != -1 ){
            switch (c) {
            case 'v':
            case 'V':
                setValidation = false;
                break;
            case 'n':
            case 'N':
                setNameSpaces = false;
                break;
            case 's':
            case 'S':
                setSchemaSupport = false;
                break;
            case '?':
            case 'h':
            case '-':
                argopt.printUsage();
                System.exit(1);
                break;
            default:
                argopt.printUsage();
                System.exit(1);
                break;
            }
        }
        return arg;
    } // parseCommandLine


    public void warning(SAXParseException spe) {
        System.err.println("[Warning] " + spe.getMessage() +
            " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
```

```
    }

    public void error(SAXParseException spe) {
        System.err.println("[Error] " + spe.getMessage() +
            " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
    }

    public void fatalError(SAXParseException spe) throws SAXException {
        System.err.println("[Fatal Error] " + spe.getMessage() +
            " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
        throw spe;
    }

} // class ParseXMLDOM
```

Here is another example to parse XML and extract key-value pairs for a dictionary from XML and Java by Hiroshi Maruyama, Kent Tamura, and Naohiko Uramoto. It uses XML documents of the form:

```
<?xml version="1.0"?>
<keyval>
  <key>URL</key>
  <value>http://www.ibm.com/xml</value>
  <key>Owner</key>
  <value>IBM</value>
</keyval>
```

```
import com.ibm.xml.parser.Parser;
import java.io.FileInputStream;
import java.io.InputStream;
import java.util.Hashtable;
import org.w3c.dom.CDATASection;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.EntityReference;
import org.w3c.dom.Node;
import org.w3c.dom.Text;

/**
 * MakeEltTblDOM.java
 **/
public class MakeEltTblDOM {

    static public void main(String[] argv) {
        if (argv.length != 1) {
            System.err.println("Missing filename.");
```

```
            System.exit(1);
        }
        try {
            // Open specified file
            InputStream is = new FileInputStream(argv[0]);

            // Start parsing
            Parser parser = new Parser(argv[0]);
            Document doc = parser.readStream(is);

            // Check if there is errors
            if (parser.getNumberOfErrors() > 0) {
                System.exit(1);
            }
            // Document is well-formed

            // Create hashtable for string key-value pairs
            Hashtable hash = new Hashtable();

            String key = null, value = null;

            // Traverse all the children of the root
            for (Node kvchild =
                doc.getDocumentElement().getFirstChild();
                kvchild != null;
                kvchild = kvchild.getNextSibling()) {

                // When child is an element
                if (kvchild instanceof Element) {
                    // If tag name is "key",
                    // store its content in vkey
                    if (kvchild.getNodeName().
                        equals("key")) {
                        key = makeChildrenText(kvchild);

                    // If tag name is "value"
                    } else if (kvchild.getNodeName().
                        equals("value")) {

                        // Extract the text content
                        value = makeChildrenText(kvchild);
                        // Check key is specified and
                        // store the key-value pair
```

```
                    if (key != null) {
                        hash.put(key, value);
                        key = null;
                    }
                }
            }
        }
        // Display the hashtable
        System.out.println(hash);

    } catch (Exception e) {
        e.printStackTrace();
    }
}

private static String makeChildrenText (Node node){
    // Create a StringBuffer to store the result.
    // StringBuffer is more efficient than String
    StringBuffer buffer = new StringBuffer();
    return makeChildrenText1 (node, buffer);
}

private static String makeChildrenText1 (
     Node node, StringBuffer buffer){

    // Visit all the child nodes
    for (Node ch = node.getFirstChild();
         ch != null;
         ch = ch.getNextSibling()) {
        // Recursive: child may have children
        if (ch instanceof Element ||
            ch instanceof EntityReference) {
            buffer.append(makeChildrenText(ch));
        // If the child is a text,
        // append it to the result buffer
        } else if (ch instanceof Text) {
            buffer.append(ch.getNodeValue());
        }
    }

    return buffer.toString();
  }
}
```

SAX

SAX is an event or notification interface for parsing. It supports DocumentHandler, DTDHandler, and ErrorHandler interfaces.

The primary events on the DocumentHandler interface are

startDocument(), **endDocument**()

startElement(String name, AttributeList atts), **endElement**(String name)

characters(char ch[], int start, int length)

ignorableWhitespace(char ch[], int start, int length)

processingInstruction(String target, String data)

Here is an example to retrieve the purchase order total using the SAX interfaces in Xerces from xml.apache.com:

```
import util.Arguments;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.io.UnsupportedEncodingException;

import org.apache.xerces.parsers.SAXParser;

import org.xml.sax.Attributes;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.ErrorHandler;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.SAXNotRecognizedException;
import org.xml.sax.SAXNotSupportedException;

public class ParseXMLSAX extends DefaultHandler {

    private static boolean setValidation    = true; //defaults
    private static boolean setNameSpaces    = true;
    private static boolean setSchemaSupport = true;

    public ParseXMLSAX() {
```

```
super();
}

public static void main(String argv[]) {

String uri = new String(parseCommandLine(argv));

    try {
        XMLReader parser = new org.apache.xerces.parsers.SAXParser();

        parser.setFeature( "http://xml.org/sax/features/namespaces", setNameSpaces );
        parser.setFeature( "http://apache.org/xml/features/validation/schema",
                           setSchemaSupport );
        parser.setFeature( "http://xml.org/sax/features/validation", setValidation );

     ParseXMLSAX parseXML = new ParseXMLSAX();
     parser.setContentHandler(parseXML);
     parser.setErrorHandler(parseXML);

     long startParse = new java.util.Date().getTime();
     parser.parse(uri);
     System.out.println("Successful SAX parse for: " + uri + " in " +
           (new java.util.Date().getTime() - startParse) + " ms.");
     System.out.println("Total: " + total_textbuf);

     } catch (org.xml.sax.SAXParseException spe) {
        System.out.println("Error: " + spe.getMessage() +
            " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
     } catch (org.xml.sax.SAXNotRecognizedException ex ){
        System.out.println("Error: " + ex.getMessage());
     } catch (org.xml.sax.SAXNotSupportedException ex ){
        System.out.println("Error: " + ex.getMessage());
     } catch (org.xml.sax.SAXException se) {
        System.out.println("Error: " + se.getMessage());
     } catch (Exception e) {
        System.out.println("Error: " + e.getMessage());
     }

  } // main(String[])

private static int state = 0;
private static int IN_TOTAL = 1;
private static StringBuffer total_textbuf = new StringBuffer();
```

```
public void startElement(String uri, String local, String raw, Attributes attrs) {
    if (local.equals("Total")) {
        state = IN_TOTAL;
        total_textbuf.setLength(0);
    }
} // startElement(String,AttributeList)

public void endElement(String uri, String local, String raw) {
        if (local.equals("Total")) {
            state = 0;
    }
 }

 public void characters(char[] ch, int start, int length)
                throws SAXException {
        if (state == IN_TOTAL) {
            total_textbuf.append(ch, start, length);
        }
 }

 public static String parseCommandLine(String argv[]) {
        Arguments argopt = new Arguments();
        argopt.setUsage( new String[] {
                            "usage: java ParseXMLSAX (options) uri ",
                            "",
                            "options:",
                            "  -v | -V  Turn off validation [default=on]",
                            "  -s | -S  Turn off Schema support [default=on]",
                            "  -n | -N  Turn off namespace [default=on]",
                            "  -h       This help screen."} );
        if (argv.length == 0) {
            argopt.printUsage();
            System.exit(1);
        }
        argopt.parseArgumentTokens(argv , new char[] { 'p'} );

        int   c;
        String arg = argopt.getlistFiles();
        while ( (c =  argopt.getArguments()) != -1 ){
            switch (c) {
            case 'v':
            case 'V':
```

```
                setValidation = false;
                break;
            case 'n':
            case 'N':
                setNameSpaces = false;
                break;
            case 's':
            case 'S':
                setSchemaSupport = false;
                break;
            case '?':
            case 'h':
            case '-':
                argopt.printUsage();
                System.exit(1);
                break;
            default:
                argopt.printUsage();
                System.exit(1);
                break;
            }
        }
        return arg;
    } // parseCommandLine


    public void warning(SAXParseException spe) {
        System.err.println("[Warning] " + spe.getMessage() +
        " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
    }

    public void error(SAXParseException spe) {
        System.err.println("[Error] " + spe.getMessage() +
        " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
    }

    public void fatalError(SAXParseException spe) throws SAXException {
        System.err.println("[Fatal Error] " + spe.getMessage() +
        " at line: " + spe.getLineNumber() + " column: " + spe.getColumnNumber());
        throw spe;
    }

} // class ParseXMLSAX
```

Communicating XML with SOAP

Any standard protocol can be used to send and receive XML. Solutions have been built with HTTP, SSL, MQSeries, MSMQ, and even EMail. The Simple Object Access Protocol (SOAP) formalizes the request and response format over HTTP to make it easy to communicate between machines without problems from firewalls. Here is a sample SOAP packet for sending an order:

```
POST /Orders HTTP/1.1
Host: www.eStore.com
Content-Type: text/xml
Content-Length: 2048
SOAPMethodName: Namespace-URI#PlaceOrder

<SOAP:Envelope xmlns:SOAP=
      "urn:schemas-xmlsoap-org:soap.v1">
   <SOAP:Body>
      <m:PlaceOrder xmlns:m="My-Namespace-URI">
               <Title>XML Handbook</Title>
               <Author>Goldfarb and Prescod</Author>
              <ShipTo>.....</ShipTo>
              <BillTo>.....</BillTo>
      </m:PlaceOrder>
   </SOAP:Body>
</SOAP:Envelope>
```

The response from the server would be something like

```
HTTP/1.1 200 OK
Connection: close
Content-Type: text/xml
Content-Length: 1024

<SOAP:Envelope xmlns:SOAP=
     "urn:schemas-xmlsoap-org:soap.v1">
   <SOAP:Body>
      <m:PlaceOrderResponse xmlns:m="My-Namespace-URI">
               <status>OK</status>
               <Delivery>7</Delivery>
      </m:PlaceOrderResponse>
   </SOAP:Body>
</SOAP:Envelope>
```

<?= indicates processing instructions

FORMATTING FOR DISPLAY

XML focuses on document structure rather than display. A style sheet provides the instructions to render that structure for a particular output device, whether it is a browser, a printer, or a voice synthesizer.

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) allow you to specify formatting for styles associated with specific tags. Examples from XML IE5 by Alex Homer:

```
<?xml version="1.0"?>
<?xml:stylesheet type="text/css" href="Corporate.css"?>
<?xml:stylesheet type="text/css" href="Department.css"?>
<?xml:stylesheet type="text/css" href="booklist.css"?>
<BOOKLIST>
 <ITEM>
  <CODE>16-048</CODE>
  <CATEGORY>Scripting</CATEGORY>
  <RELEASE_DATE>1998-04-21</RELEASE_DATE>
  <TITLE>Instant JavaScript</TITLE>
  <SALES>375298</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-105</CODE>
  <CATEGORY>ASP</CATEGORY>
  <RELEASE_DATE>1998-05-10</RELEASE_DATE>
  <TITLE>Instant Active Server Pages</TITLE>
  <SALES>297311</SALES>
 </ITEM>
 <ITEM>
  <CODE>16-041</CODE>
  <CATEGORY>HTML</CATEGORY>
  <RELEASE_DATE>1998-03-07</RELEASE_DATE>
  <TITLE>Instant HTML</TITLE>
  <SALES>127853</SALES>
 </ITEM>
</BOOKLIST>
```

Booklist.css

```
ITEM
{
    DISPLAY: block;
    MARGIN: 15px
}
CODE
{
    DISPLAY: inline;
    FONT-FAMILY: Tahoma,Arial,sans-serif;
    FONT-SIZE: 10pt;
    FONT-WEIGHT: bold
}
CATEGORY
{
    COLOR: darkgray;
    DISPLAY: inline;
    FONT-FAMILY: Tahoma,Arial,sans-serif;
    FONT-SIZE: 12pt;
    FONT-WEIGHT: bold
}
RELEASE_DATE
{
    COLOR: red;
    DISPLAY: inline;
    FONT-FAMILY: Tahoma,Arial,sans-serif;
    FONT-SIZE: 10pt
}
TITLE
{
    BACKGROUND-COLOR: black;
    COLOR: white;
    DISPLAY: inline;
    FONT-FAMILY: Tahoma,Arial,sans-serif;
    FONT-SIZE: 12pt
}
SALES
{
    DISPLAY: none
}
```

Transformations via Style Sheets – XSLT

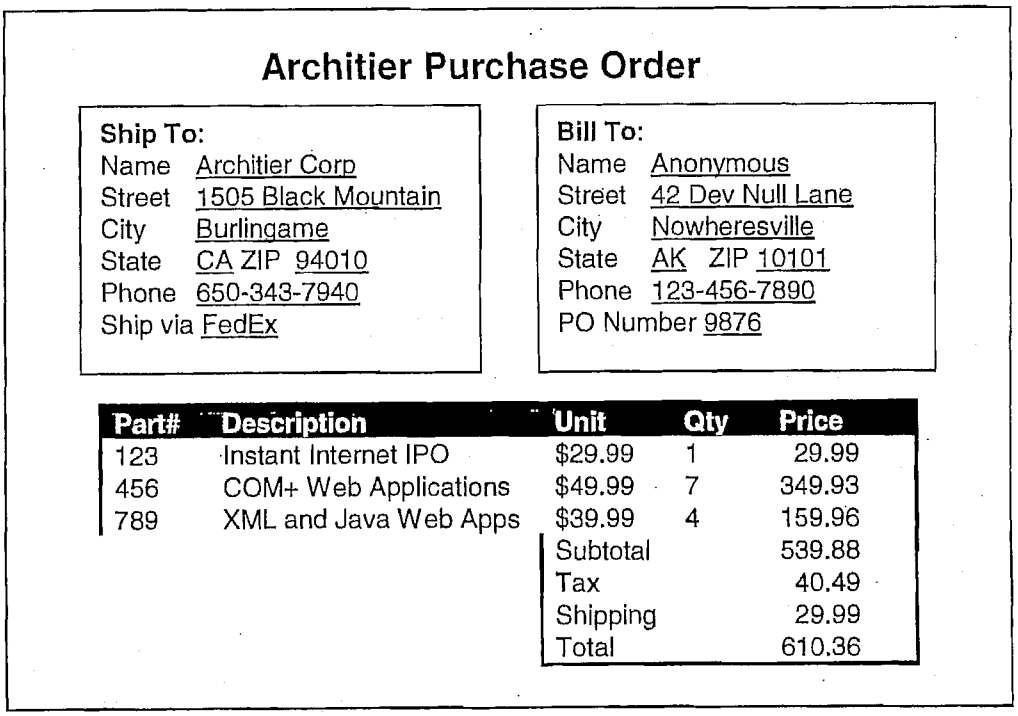
Instead of simply displaying formatting for single elements at a time, XSLT allows complete transformations on the document. Conceptually an input document tree is transformed into a different output tree, potentially changing the language, the element arrangement, or element attributes.

A pattern or filter is used to identify a set of nodes in the source tree. A template is applied to the matching nodes to produce the output nodes.

The most broadly distributed stylesheet engine is built into IE5. Unfortunately, it uses a working draft version of the standard—not the final November 1999 specification.

Example

We start with a sample purchase order document and its associated XML file. Then we will look at two very simple transforms that produce HTML output to see how a transform engine works.



Architier Purchase Order

Ship To:
Name Architier Corp
Street 1505 Black Mountain
City Burlingame
State CA ZIP 94010
Phone 650-343-7940
Ship via FedEx

Bill To:
Name Anonymous
Street 42 Dev Null Lane
City Nowheresville
State AK ZIP 10101
Phone 123-456-7890
PO Number 9876

Part#	Description	Unit	Qty	Price
123	Instant Internet IPO	$29.99	1	29.99
456	COM+ Web Applications	$49.99	7	349.93
789	XML and Java Web Apps	$39.99	4	159.96
		Subtotal		539.88
		Tax		40.49
		Shipping		29.99
		Total		610.36

Example 1 PO Total

The purchase order total is: **$610.36**

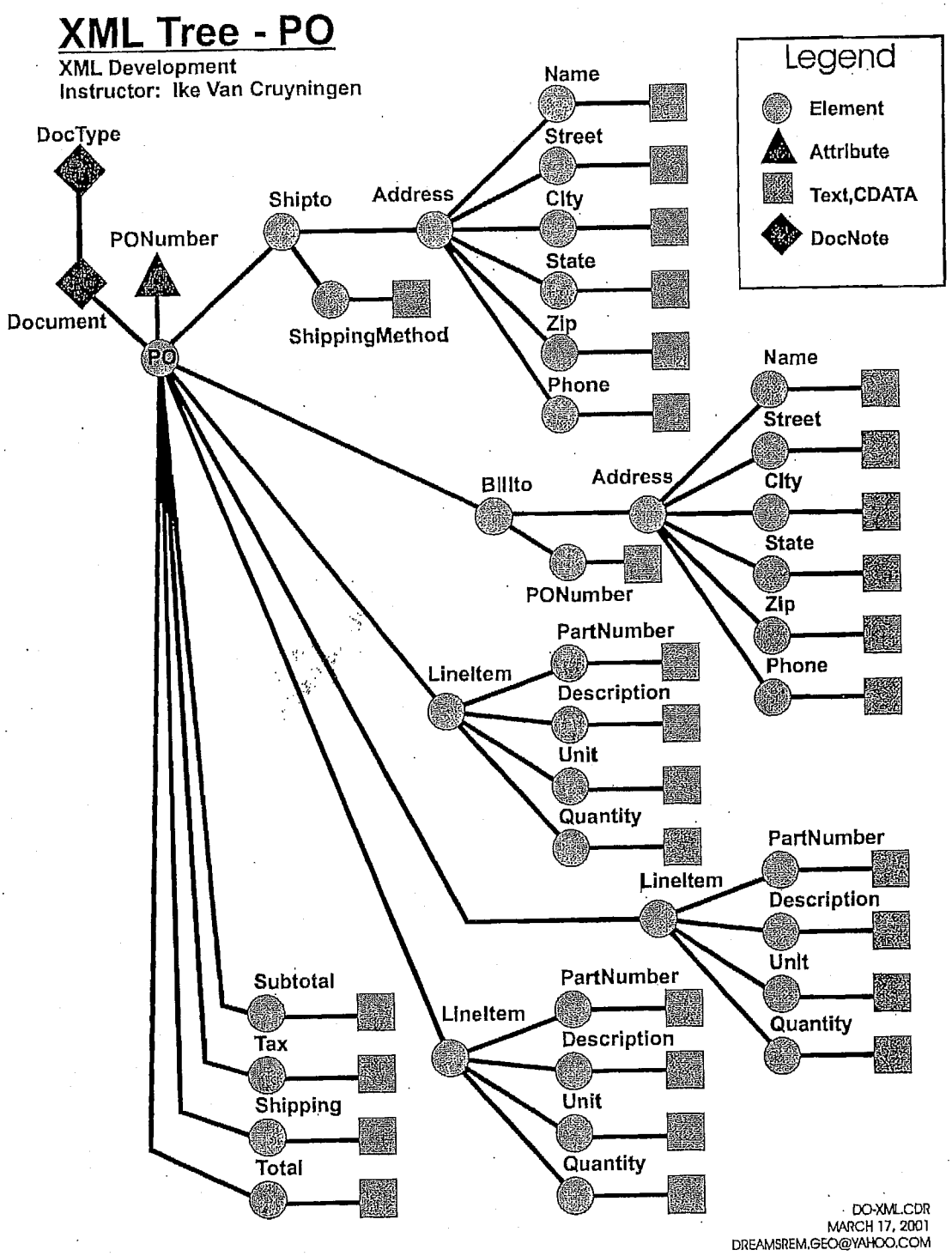
Example 2 PO Line Items

Part#	Description	Unit	Qty	Ext Price
123	Instant Internet IPO	$29.99	1	29.99
456	COM+ Web Applications	$49.99	7	349.93
789	XML and Java Web Apps	$39.99	4	159.96

PO.xml

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="POTotal.xsl"?>
<PO xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation="POTotal.xsd"
   PONum="9876">
 <ShipTo>
   <Address>
      <Name>Architier Corp</Name>
      <Street>1505 Black Mountain</Street>
      <City>Burlingame</City>
      <State>CA</State>
      <Zip>94010</Zip>
      <Phone>650-343-7940</Phone>
   </Address>
   <ShipMethod>Fed Ex</ShipMethod>
 </ShipTo>
 <BillTo>
   <Address>
      <Name>Anonymous</Name>
      <Street>42 Dev Null Lane</Street>
      <City>Nowheresville</City>
      <State>AK</State>
      <Zip>10101</Zip>
      <Phone>123-456-7890</Phone>
```

```
  </Address>
  <PONumber>9876</PONumber>
</BillTo>
<LineItem>
  <PartNumber>123</PartNumber>
  <Description>Instant Internet IPO</Description>
  <UnitPrice>29.99</UnitPrice>
  <Quantity>1</Quantity>
</LineItem>
<LineItem>
  <PartNumber>456</PartNumber>
  <Description>XSLT Programmers Reference</Description>
  <UnitPrice>49.99</UnitPrice>
  <Quantity>7</Quantity>
</LineItem>
<LineItem>
  <PartNumber>789</PartNumber>
  <Description>XML &amp;amp; Java Web Apps</Description>
  <UnitPrice>39.99</UnitPrice>
  <Quantity>4</Quantity>
</LineItem>
<SubTotal>539.88</SubTotal>
<Tax>40.49</Tax>
<Shipping>29.99</Shipping>
<Total>610.36</Total>
</PO>
```


XML Tree - PO
XML Development
Instructor: Ike Van Cruyningen
Legend
Element
Attribute
Text,CDATA
DocNote
DocType
Document
PONumber
PO
Shipto
Address
Name
Street
City
State
Zip
Phone
ShippingMethod
BillTo
Address
Name
Street
City
State
Zip
Phone
PONumber
LineItem
PartNumber
Description
Unit
Quantity
LineItem
PartNumber
Description
Unit
Quantity
Subtotal
Tax
Shipping
Total
LineItem
PartNumber
Description
Unit
Quantity
DO-XML.CDR
MARCH 17, 2001
DREAMSREM.GEO@YAHOO.COM

To pull Attribute
<xsl:apply-templates select="/PO/@PONumber" />
<xsl:template match="@PONumber">
<xsl:value-of select="." />
</xsl:template>

POTotal.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">  root
   <html><head><title>PO Total</title></head>
   <body>
   The purchase order total is:
   <xsl:apply-templates select="/PO/Total" />  or <xsl:value-of select="/PO/Total" />
   </body>
   </html>
</xsl:template>

<xsl:template match="Total">  current node
   <b>$<xsl:value-of select="." /></b>
</xsl:template>

</xsl:stylesheet>
```

POLineItemTable.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl=
"http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="/">
   <HTML><HEAD><TITLE>PO Line Items</TITLE></HEAD>
   <BODY>
   <TABLE>
   <TR><TH>Part Number</TH><TH>Description</TH>
      <TH>Unit Price</TH><TH>Quantity</TH>
      <TH>Extended Price</TH></TR>
   <xsl:apply-templates select="/PO/LineItem">
      <xsl:sort select="Description" order="ascending" />
   </xsl:apply-templates>
   </TABLE>
   </BODY>
   </HTML>
</xsl:template>

<xsl:template match="LineItem">
   <tr>
```

```
      <td align="center">
         <xsl:value-of select="PartNumber" /></td>
      <td align="center">
         <xsl:value-of select="Description" /></td>
      <td align="center">
         <xsl:value-of select="Quantity" /></td>
      <td align="right">
         <xsl:value-of select="UnitPrice" /></td>
      <td align="right">
         <xsl:value-of select="format-number(
            UnitPrice * Quantity, '$#,###.##')" /></td>
   </tr>
</xsl:template>

</xsl:stylesheet>
```

XPath

XPath is a tool to identify nodes in an XML tree. Think of it as a general search tool for trees.

Data Structure	Search Tool
String	Regular Expression
Relational Database	SQL select statement
XML Tree	XPath

Both XSLT and XLink need a way of identifying nodes in an XML tree. XPath is the standard used in both these applications. XPath models an XML document as a tree of nodes. These include element, attribute, text, and other types of nodes. The primary XPath construct is an expression, which results in a node-set, a boolean, a number, or a string.

Location Steps

In XSLT applications the XPath expression identifies a number of nodes in a tree. The expression looks much like a directory path with 'location steps' representing different levels in the hierarchy.

/ [location step] / [location step] / [location step]

A location step is a combination of an axis, node test, and predicate.

/ axis::nodetest[predicate] / axis::nodetest[predicate]

The axis is the tree relationship, the node test specifies the node type or name, and the predicate refines the set of nodes. In child::para[position()=1], child is the name of the axis, para is the node test and [position()=1] is a predicate

The abbreviated axis forms are:

// recursively searches branch for all matches // wildcard across generations/level

* matches any element at a single level * wildcard for one generation/level

@ matches an attribute

. matches the current context

.. parent directory /PO/* will get all the children

/PO/*/* will get all the grandchildren

The axis specifies a direction in the XML tree. The default axis is 'child', but you have many other options.

AxisName ::= 'child' | 'descendant' | 'descendant-or-self' |

'parent' | 'ancestor' | 'ancestor-or-self' |

'preceding-sibling' |'following-sibling' |

'preceding' | 'following' |

'self' | 'attribute' | 'namespace'

Predicate

The predicate or filter limits the selection of nodes to those that meet that criterion. It selects a subset of all the nodes that meet the path expression, much like a WHERE clause in SQL limits the results from those that meet the SELECT clause. It is a sub-query in square brackets []. It searches through nodes to match a criterion. It can check for

- existence of nodes: [ElementTypeName]

```
CatalogItem[sales]
```

- values of nodes: [ElementTypeName = 'Value']

```
CatalogItem[sales > 10000]
CatalogItem[name = 'Investing in the face of the bear']
```

- existence of attributes: [@AttributeName]

```
CatalogItem[@shipDate]
```

- values of attributes: [@AttributeName = 'Value']

```
CatalogItem[@shipDate > '2001-09-01']
```

- position of elements: [1] to [last()]

```
bullet[1]
bullet[position() > 1 and position() < last()]
bullet[position() = last() - 1]
bullet[position() = last()]
```

- /PO//address/ZIP[. > 90000] all zip codes with value greater than 90000
- /PO//address[ZIP> 90000] all Addresses with zip codes greater than 90000
- /PO//address[ZIP > 90000]/ZIP all zip codes where Addresses contain zip codes where the zip code is greater than 90000

Examples

- chapter[title="Introduction"] selects the chapter children of the context node that have one or more title children with string-value equal to Introduction
- `child::chapter/descendant::para` selects the `para` element descendants of the `chapter` element children of the context node
- `child::*/child::para` selects all `para` grandchildren of the context node
- `descendant-or-self::para` selects the `para` element descendants of the context node and, if the context node is a `para` element, the context node as well
- employee[@secretary and @assistant] selects all the employee children of the context node that have both a secretary attribute and an assistant attribute
- `/child::doc/child::chapter[position()=5]/child::section[position()=2]` selects the second `section` of the fifth `chapter` of the `doc` document element
- `child::para[attribute::type='warning'][position()=5]` selects the fifth `para` child of the context node that has a `type` attribute with value `warning`
- `child::*[self::chapter or self::appendix][position()=last()]` selects the last `chapter` or `appendix` child of the context node
- anils

XSLT

Top-level Nodes

<xsl:stylesheet> and <xsl:transform> are synonyms for the root of the XSLT document

<xsl:include href="url" /> incorporate template definitions from another file. Any duplicate declarations will result in processing errors

<xsl:import href="url" /> incorporate templates definitions from another file at a lower priority than the ones in the current file. Duplicate declarations do not result in errors, but are resolved by the priority mechanism.

<xsl:output method="xml | html | text" .../> Specifies output type and top-level elements. This allows you to relax or turn off syntax checks on the output tree.

Template Definitions

You define templates with the following syntax. The attributes and xsl:param are optional, but you must have either a name or a match.

```
<xsl:template match="Pattern" name="MyTemplate" priority="2" mode=TOC">
    <xsl:param name="MyParam" select="DefaultValueExpression" />
    template body
</xsl:template>
```

Invoking a template

Within a template body you apply new templates in one of two ways:

```
<xsl:apply-templates select="NodeExpression" mode="TOC" >
    <xsl:with-param name="MyParam" select="ParamValue" />
    <xsl:sort select="SortKeyExpression"
        order="ascending | descending"
        data-type="text | number"
```

```
        lang="collatingLanguage"
        case-order="upper-first | lower-first" />
</xsl:apply-templates>

<xsl:call-template name="MyTemplate">
    <xsl:with-param name="MyParam" select="ParamValue" />
</xsl:call-template>
```

Generating output

To generate output nodes you could just use nodes in the template body, e.g.

```
<HTML><HEAD><TITLE>XSLT Example</TITLE></HEAD><BODY>
```

This approach is easier to write and understand. However, there are special cases (whitespace issues and attributes) where it is hard to get exactly what you want on the output. Then you can use the <xsl:element>, <xsl:attribute>, <xsl:comment>, <xsl:processing-instruction>, and <xsl:text> to generate output nodes.

<xsl:value-of select="nodeExpression" /> makes it easy to extract the node value.

<xsl:copy> copies the current node to the output tree.

<xsl:copy-of select="NodeExpression" /> copies a tree fragment or node set to the output. This is a deep copy.

Programming Language Features

<xsl:if test="Expression"> ... </xsl:if> provides conditional execution or output. Note there is no <xsl:else>, so have to use

```
<xsl:choose>
    <xsl:when test="expr"> ... </xsl:when>
    <xsl:otherwise> ... </xsl:otherwise>
```

</xsl:choose>

You loop over a set of nodes with

<xsl:for-each select="Expression">

    <xsl:sort select="" />

    template body

</xsl:for-each>

You define a constant with

<xsl:variable name="city" select="'San Francisco'" />

or

<xsl:variable name="city">San Francisco</xsl:variable>

and refer to that constant within expressions with $city

PurchaseOrder.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0">

<xsl:template match="/">
   <html>
   <body>
      <img src="ArchitierSmall.gif" />
      <h3 align="center">Architier Purchase Order</h3>
      <table width="100%">
      <tr><td width="50%">
         <table width="100%">
         <xsl:apply-templates select="/PO/ShipTo" />
         </table>
      </td><td width="50%">
```

```
        <table width="100%">
        <xsl:apply-templates select="/PO/BillTo" />
        <xsl:apply-templates select="/PO/@PONumber" />
        </table>
      </td></tr>
      </table>
      <p /><hr />

      <table style="width:100%;">
         <tr>
            <th align="center">Part Number</th>
            <th align="center">Description</th>
            <th align="center">Quantity</th>
            <th align="right">Unit Price</th>
            <th align="right">Extended Price</th>
         </tr>
         <xsl:apply-templates select="PO/LineItem" >
         <xsl:sort select="PartNumber" order="ascending" />
         </xsl:apply-templates>
      </table>
      <hr />

      <table style="width:100%;">
      <tr><td align="right" width="92%">SubTotal: </td>
      <td align="left"><b><xsl:value-of
         select="/PO/SubTotal" /></b></td></tr>
      <tr><td align="right" width="92%">Tax: </td>
      <td align="left"><b><xsl:value-of
         select="/PO/Tax" /></b></td></tr>
      <tr><td align="right" width="92%">Shipping: </td>
      <td align="left"><b><xsl:value-of
         select="/PO/Shipping" /></b></td></tr>
      <tr><td align="right" width="92%">Total: </td>
      <td align="left"><b><xsl:value-of
         select="/PO/Total" /></b></td></tr>
      </table>
   </body>
   </html>
</xsl:template>

<xsl:template match="ShipTo">
   <tr><th align="center" colspan="2">Ship To</th></tr>
   <xsl:apply-templates select="Address" />
```

```
   <tr><td align="right">Ship Method: </td>
   <td><b><xsl:value-of
      select="ShipMethod" /></b></td></tr>
</xsl:template>

<xsl:template match="BillTo">
   <tr><th align="center" colspan="2">Bill To</th></tr>
   <xsl:apply-templates select="Address" />
</xsl:template>

<xsl:template match="Address">
   <tr><td align="right">Name: </td>
   <td><b><xsl:value-of select="Name" /></b></td></tr>
   <tr><td align="right">Street: </td>
   <td><b><xsl:value-of select="Street" /></b></td></tr>
   <tr><td align="right">City: </td>
   <td><b><xsl:value-of select="City" /></b></td></tr>
   <tr><td align="right">State: </td>
   <td><b><xsl:value-of select="State" /></b></td></tr>
   <tr><td align="right">Zip: </td>
   <td><b><xsl:value-of select="Zip" /></b></td></tr>
   <tr><td align="right">Phone: </td>
   <td><b><xsl:value-of select="Phone" /></b></td></tr>
</xsl:template>

<xsl:template match="@PONumber">
   <tr><td align="right">PO Number</td>
   <td><b><xsl:value-of select="." /></b></td></tr>
</xsl:template>

<xsl:template match="LineItem">
   <tr>
      <td align="center"><xsl:value-of
         select="PartNumber" /></td>
      <td align="center"><xsl:value-of
         select="Description" /></td>
      <td align="center"><xsl:value-of
         select="Quantity" /></td>
      <td align="right"><xsl:value-of
         select="UnitPrice" /></td>
      <td align="right"><xsl:value-of
 select="format-number(UnitPrice * Quantity, '$#,###.##')"
/></td>
```

```
    </tr>
</xsl:template>

</xsl:stylesheet>
```

LINKING DOCUMENTS—XLL AND XLP

Links between documents are an integral part of the WWW. They make it a 'web'. You link to another document with the anchor tag

```
<a HREF = "http://www.destinationserver.com"
   TITLE = "Title to display while getting document">
   Underlined text displayed in browser</a>
```

You can also name part of a document and refer to that part in the URL.

```
<a NAME = "Section3">Section 3</a>
      <!-- define a name in the destination document -->
```

```
See <a   HREF = "#section3">Section 3</a>
      <!-- use the name in the link -->
```

Note the target document author has to name the section before you can refer to it, so you need to cooperate.

HTML links work well, but they are relatively primitive compared to other hypertext systems. Limitations of HTML links are:

- Links are embedded in source documents. Independent authors or commentators cannot add links to other destinations without modifying the source documents.
- To link to a section of a target document it has to be explicitly named in the target. The target document author has to get involved to define the section.
- Links should point to parts of the document structure, rather than named anchors. The content will change more rapidly than the structure.

- Source links can only point to one other document at a time, rather than a list of related links
- You cannot provide additional attributes for links. You would like to supply different types of links like explanations, commentary, critiques, or previous versions. You might also want to distinguish private from public links.

When a user clicks an HTML hyperlink, the file being linked to replaces the current Web page. XLink lets Web builders add behaviors to links. Today, for example, you have to use a bit of JavaScript to make a link pop up a separate window, but XLink lets Web builders code links to perform a variety of actions, including popping up a menu of linking choices.

XLink supports links that are external to the documents being linked

XLinks can point to more than one resource

Simple Links

Simple links are like the HTML <A> tag with a single target and one-way traversal. (from Proposed Recommendation 20 December 2000)

```
<mylink
   xmlns:xlink="http://www.w3.org/1999/xlink"
   xlink:type="simple"
   xlink:href="mydoc.xml" <!-- target URI -->
   xlink:show="replace"
      <!-- new|replace|embed|other|none -->
   xlink:actuate="onRequest"
      <!-- onLoad|onRequest|other|none -->
   xlink:role="urn:roles-architier-com:author"
      <!--  relationships between docs URI! -->
   xlink:title="caption"
      <!-- text display for user -->
   >
   See my document
</mylink>
```

A show=embed along with actuate=onLoad creates a compound document that always pulls the latest information from a variety of sources.

Adding an ID to every element in your document makes it easy to add external links. Alternatively, as in HTML, you create or mark an element with a role to make it an easy destination for a link (XPointers described below provide even more options):

```
<myanchor xml:link="simple" role="Ch2_Start" >
   <Chapter>2</Chapter>
</myanchor>
```

Extended Links

Extended links allow several targets with a relationship or attributes for each target. You need to distinguish the link (which is like a relation is a relational database) and the locators (which are the addresses the link points to). Just as a many-to-many mapping in a database requires a separate table, extended links use a list of locators instead of a single HREF. Here is an example of a list of related topics that could be displayed as alternatives in a pop-up menu:

```
<mylink xml:link="extended" inline="false">
   <mytarget xml:link="locator" inline="false"
      role="Summary" title="Summary"
      href="help/summary.xml" />
   <mytarget xml:link="locator" inline="false"
      role="Details" title="Detailed help"
      ref="help/details.xml" />
   <mytarget xml:link="locator" inline="false"
      role="Examples" title="Sample Usage"
      ref="help/examples.xml" />
   <mytarget xml:link="locator" inline="false"
      role="Glossary" title="Definition"
      ref="help/glossary.xml#/Def/keyword" />
</mylink>
```

Here is an example of a group of documents that can be retrieved in one step:

```
<mylink xml:link="group" inline="false">
   <mytarget xml:link="document" inline="false"
      href="Chapter1.xml" />
   <mytarget xml:link="document" inline="false"
      href="Chapter2.xml" />
   <mytarget xml:link="document" inline="false"
```

```
      href="Chapter3.xml" />
</mylink>
```

XPointer

In HTML it's possible to link to the middle of a page only if the author of that page put an anchor tag there. With XPointer you can "address to" (not "link to") any part of someone else's text. It's easy to see how this would be useful in working with business, legal, scientific, and academic papers.

Goal is to be able to refer to document fragments without having to change the resource containing the fragment. Syntax is

www.mycompany.com/mydocument.xml#xpointer to download whole document

www.mycompany.com/mydocument.xml|xpointer to download fragment

Xpointer is a combination of an absolute starting point in the document followed by relative locations.

An "address" consists of a URI (URL or URN), followed by an XPath, and optional addressing within an element, such as a specific character within a text node.

XPointer extends XPath with

- two new axes to model user selection: range::, which addresses a range such as a user selection by addressing its start and end; and string::, which addresses a range by string matching. Such results correspond to the range construct in the Document Object Model (DOM) rather than to specific nodes, and therefore require different treatment by applications. They are crucial, however, for representing extremely common cases anticipated for XPointer usage: locating endpoints of user-created hyperlinks, which are commonly general user selections.
- new absolute location paths here() and origin(), to provide for addressing relative to the location of an XPointer expression itself, and to the point of origin for hypertext traversal when XPointers are used in that (very common) application domain. These functions in effect add information to the context of evaluation.
- a predicate function unique(), described below, to enable testing whether an XPointer expression or sub-expression locates a single node rather than multiple nodes or no nodes.

A range is simply the combination of two XPointers:

```
id("a23")/range::child[1],following-sibling[2]
```

In the Candidate Recommendation (Last Call Working Draft 8 January 2001), this has changed to:

location-set **range-to**(location-set)

```
id("a23")/child[1]/range-to(following-sibling[2])
```

A string allows selection of a range of characters:

```
string::instance, search string,
   starting character position for selection,
   number of characters to select
```

In the Candidate Recommendation (Last Call Working Draft 8 January 2001), this has changed to:

location-set **string-range**(location-set, string, number?, number?)

For example, the following XPointer selects the range surrounding "yn" in the 3rd occurrence of the string "Thomas Pynchon" in a P element.

```
string-range(//P, "Thomas Pynchon", 9, 2)[3]
```

Or, alternatively, this could be written as:

```
string-range(string-range(//P, "Thomas Pynchon")[3], "yn",
1, 2)
```

For example, the following XPointer selects the position immediately preceding the letter "P" (8 from the start of the string) in the third occurrence of the string "Thomas Pynchon":

```
string::3,"Thomas Pynchon",8
```

The following XPointer selects the fifth exclamation mark and the character immediately following it:

```
string::5,'!',1,1
```

For purposes of string matching, the text of the element means all the character data in the element(s) in the current location source and descendant elements.

Markup characters are ignored. Thus a match may still occur when markup intervenes. For example,

```
string::1,'affect'
```

will match at a point in the document structure corresponding to XML input such as

```
<corr sic='e'>a</corr>ffect
```

Sequences of whitespace characters are treated as equivalent to a single space character.

SUMMARY

File Types:

HTML for browser display

HTML Forms for collecting user input

XML for content

DTD or Schema for structure and validation

XSLT for transformations

CSS and XSL FO for detailed formatting

XPath for identifying nodes

XLink and XPointer for references between documents.

Tools:

Any app server technology for generating XML (e.g. ASP, JSP, servlets, ColdFusion, CGI Programs, etc.)

DOM or XSL for tree manipulation

SAX for searching

Further Questions

References and Web sites listed at start of notes

www.architier.com

ikevc@architier.com

ASSIGNMENT A

Due at the start of next class. Print out your results and bring them to class.

1) (10) Create an XML document containing the data in this purchase order:

Architier Purchase Order

Ship To:
Name Architier Corp
Street 1505 Black Mountain
City Burlingame
State CA ZIP 94010
Phone 650-343-7940
Ship via FedEx

Bill To:
Name Anonymous
Street 42 Dev Null Lane
City Nowheresville
State AK ZIP 10101
Phone 123-456-7890
PO Number 9876

Part#	Description	Unit	Qty	Price
123	Instant Internet IPO	$29.99	1	29.99
456	COM+ Web Applications	$49.99	7	349.93
789	XML and Java Web Apps	$39.99	4	159.96
		Subtotal		539.88
		Tax		40.49
		Shipping		29.99
		Total		610.36

Use elements type names Name, Street, City, State (one of WA, OR, CA, NV, AK), Zip, Phone, ShipMethod (one of UPSGround, UPSBlue, UPSRed, FedEx, USMail), PONumber, PartNumber, Description, Quantity, UnitPrice,
(Don't include Extended Price = Unit Price * Quantity : this will be calculated in the stylesheet) SubTotal, Tax, Shipping, and Total.

2) (15) Create the simplest possible DTD for the purchase order.

3) (5) Look up a purchase order DTD or Schema on the Web, print out the URL and core of the definition, and compare it to your DTD. If you send me an email asking how to find something on the Web, you get zero points.

Assignment B

Due ten days from the last class via email to **ucbxml@architier.com**

Do not send attachments, bitmaps, or URLs! You will get an automated response confirming receipt of your email. If you do not get this response, your assignment has not been submitted. Then print it out and express mail it to the Ship To address.

4) (20) With your favorite programming language and an XML parser, use the DOM to manipulate the purchase order document. By iterating over the unit prices and quantities, calculate the item extended prices, subtotal, and total prices. Print out (text/plain or text/html) a table showing only the part numbers, extended prices, and total. Do not use XSL!

5) (25) Download either http://www.architier.com/courses/XML/Po.zip or http://www.architier.com/courses/XML/Poms.zip (Microsoft) for the purchase order examples. The stylesheets display the purchase order in HTML by transforming the PO.xml file.

Please create two new stylesheets to create HTML for a Packing List and for an Invoice using the PO.xml data. Assume the second line item is not in stock so it will not be shipped, nor invoiced. Don't worry if the subtotal and total are incorrect in this question—you'll fix them in the next question.

6) (25) Look in the help pages for your stylesheet renderer to figure out how to calculate values in a stylesheet (most of them support an extension language or recursion). Then replace the subtotal and total stylesheet elements in question 5) with calculations for the sum of the extended prices.

OR

(20) Develop a style sheet to read the purchase order on a voice synthesizer using the CSS Aural styles with the items arranged in alphabetical order of the Description field. (5) Find a parser-to-synthesizer connection to read the purchase order on your computer.

Please turn in **one email** containing the code/script and the result table for question 4) and the style sheets for questions 5) and 6). **Do not send attachments, bitmaps, or URLs.**