Issue Date: 30-06-2000		**Version Number: 1.0**

XML ASSESSMENT USAGE REPORT

Written by:

Dimitrios Oikonomidis & Michael Papaioannou,

Students of Department of Informatics at

Athens University of Economics and Business

Table of Contents

1. INTRODUCTION

This report has been written as a result of our co-operation with European Dynamics S.A. during our Practical Exercise in the Spring term of the academic year 1999-2000 of our studies at the Athens University of Economics and Business. It negotiates about a new standard in the computer world, XML, the eXtensible Mark-up Language. As its name suggests, XML is another mark-up language after SGML and HTML, with whom shares a lot of common features, but also is quite different and more powerful in Internet-working, due to its special capabilities, which will be discussed analytically in this essay.

In this essay, we have given special emphasis on:

- The way XML supports structured queries in order to extend the range of search options;
- The presentation of the same document (content) with different views;
- The customization of presentation based on profiling information; and
- The use of XML as a universal hub for data interchange between heterogeneous systems over the Web.

In **Chapter 2**, we are giving a general image about XML and why it was constructed in the first place. A brief presentation of its syntax is also included, alongside with some discussion about DTDs, Links, APIs and other useful terms of the language. Furthermore, we are having a look into the usage of XML today as well as its future and especially its integration with EDI.

In **Chapter 3,** we are dealing with the topic of the presentation of XML. Some of the aspects discussed here are the options for displaying XML documents, why we should use a style sheet, how it works, an introduction to XSL, the Style sheet Language for XML, and its structure as well as a variety of examples of its use.

Chapter 4: Intelligent querying of XML documents appears to be necessary in order to locate and obtain data which are stored in them. The XML Query Language, XQL, is the right tool to achieve the above target. It works in a way similar to the Structured Query Language, SQL, but is applied over XQL documents and has a variety of operators that are presented in this chapter using examples.

In **Chapter 5** the topic that is discussed has to do with Internet Searching today, using XML. XML metadata is the revolution in the field of Internet Searching and promises better and accurate results. This chapter shows how this can be achieved. Moreover, an implementation of an XML searching engine, the GoXML, is presented as an example.

In **Chapter 6** XML is presented as an intermediate means of exchanging data between databases of different structure. Examples are being given for the modeling of data from relational databases to XML documents and vice versa. Also how data originated from an object-oriented database can be represented in an XML document. In addition, the best management system for the storing and retrieving of XML documents is suggested, alongside with a presentation of XML Utilities for PL/SQL in Oracle and XML support in JDeveloper.

In **Chapter 7** an extension of the HTTP (Hyper Text Transfer Protocol) is presented. WebDAV is a cutting edge technology which promises to make it easier to perform remote collaborative project work over the web, and it is based on XML. It extends the web to make authoring of web resources as easy as browsing them. As it is said, using WebDAV ''It's easier than ever before to assemble a virtual team for remote collaborative project work''. This chapter is an introduction on WebDAV providing a lot of information necessary for anyone who is interested in it.

In **Chapters 8 and 9** we present two of the most significant metadata formats available in XML for description and presentation of information in the web: RSS (Rich Site Summary) and RDF (Resource Description Format). What are discussed here are their syntax and features, their usage as well as some tools available, and predictions for their future.

In **Appendix A,** a comparison between XML and other mark-up languages is covered. XML is compared with SGML and HTML regarding their features, their ease in use and their possibilities in addition to limitations.

In **Appendix B**, we include a chart displaying the various levels of XML support in the last-generation browsers.

We feel the obligation to thank the people of European Dynamics with whom we came in contact during our co-operation, for giving us the opportunity to have the experience of working in a major software enterprise and for their useful guidance and comments during the edition of this essay - Dr. Dimitrios Gritsis, Director in the Division of International Marketing and Sales, Dr. Isidore Kounoupas and Dr. Panagiotis Retzepopoulos from the Software Division of the company. Last, but not least, we would also like to thank our Professor, Dr. Michail Vazirgiannis, Lecturer in the Department of Informatics at the Athens University of Economics and Business, for his valuable support and advice.

The authors,

Dimitris Oikonomidis

Michael Papaioannou

2. XML

2.1 Introduction in XML

The eXtensible Markup Language (XML) is a subset of SGML (Standard Generalized Mark-up Language). Its goal is to enable generic SGML to be served, received, and processed on the Web in the way that is now possible with HTML. XML has been designed for ease of implementation and for interoperability with both SGML and HTML [REC98].

XML was developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (**W3C**) in 1996. The current XML specification of W3C (www.w3.org) is v1.0 and is dated in 10 February 1998. XML is a markup language for documents containing structured information. A markup language is a mechanism to identify structures in a document. Structured information contains both content and some indication of what role that content plays.

However XML is not a single, predefined mark-up language. XML defines a standard way to add mark-up to documents without specifying either semantics or a tag set. In fact XML is really a meta-language for describing mark-up languages. In other words, XML provides a facility to define tags and the structural relationships between them. Since there's no predefined tag set, there can't be any preconceived semantics. All of the semantics of an XML document will either be defined by the applications that process them or by style sheets.

XML is primarily intended to meet the requirements of large-scale Web content providers for industry-specific mark-up, vendor-neutral data exchange, media-independent publishing, one-on-one marketing, workflow management in collaborative authoring environments, and the processing of Web documents by intelligent clients. The language is designed for the quickest possible client-side processing consistent with its primary purpose as an electronic publishing and data interchange format (Press Release of W3C). XML has been developed in order to address the requirements of commercial Web publishing and enable the further expansion of Web technology into new domains of distributed document processing, offering augmented functionality in comparison with the current Hypertext Mark-up Language (HTML). [BOS97]

2.2 Design goals for XML

According to the official W3C XML 1.0 specification (10-2-98) the design goals for XML are the following, accompanied with a slight interpretation of them:

1. XML should be straightforwardly usable over the Internet, i.e. users must be able to view XML documents as quickly and easily as HTML documents.
2. XML should support a wide variety of applications, like authoring, browsing, content analysis, etc.
3. XML should be compatible with SGML, i.e. XML was designed pragmatically to be compatible with existing standards while solving the relatively new problem of sending richly structured documents over the web.
4. It should be easy to write programs which process XML documents, i.e. it ought to take about two weeks for a competent computer science graduate student to build a program that can process XML documents.

5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero, since optional features inevitably raise compatibility problems and sometimes lead to confusion and frustration.
6. XML documents should be human-legible and reasonably clear, i.e. users ought to be able to look at an XML document in a text editor if they don't have an XML browser and actually figure out what the content means.
7. The XML design should be prepared quickly, i.e. XML was needed immediately and was developed as quickly as possible.
8. The design of XML should be formal and concise, i.e. XML must be expressed in **EBNF** (Extended Backus-Naur Form) and must be amenable to modern compiler tools and techniques.
9. XML documents should be easy to create, i.e. it must be possible to create XML documents directly in a text editor, with simple shell and Perl scripts, etc.
10. Terseness in XML markup is of minimal importance, i.e. features of SGML language designed to minimize the amount of typing required to manually key in SGML documents are not supported in XML.

2.3 Syntax of XML

XML describes a class of data objects called XML documents and partially describes the behaviour of computer programs which process them. XML documents are made up of storage units called *entities*, which contain either parsed or unparsed data. Parsed data is made up of *characters*, some of which form the *character data* (content) in the document, and some of which form *mark-up*. Mark-up encodes a description of the document's storage layout and logical structure. [REC98]

An example of an XML file is the following:

```
<?xml version="1.0" standalone=“yes" encoding=“UTF-8"?>
<!– Example of a simple XML file -->
<example>
  <student code=“p3960134”>
     <name>
       <firstname>John</firstname>
       <lastname>Doe</lastname>
     </name>
     <department year=“4”>Informatics</department>
     <address>
       <street>Patision 238</street>
       <city zipcode=“12356”>Athens</city>
     </address>
  </student>
</example>
```

There are six kinds of markup that can occur in an XML document: elements, entity references, comments, processing instructions, marked sections, and document type declarations.[WALS98]

- **Elements,** delimited by angle brackets, identify the nature of the content they surround. Some elements may be empty in which case they have no content. If an element is not empty, it begins with a start-tag, `<element>`, and ends with an end-tag, `</element>`. For example:
  `<street>Patision 238</street>.`
  Attributes are name-value pairs that occur inside start-tags after the element name. For example:
  `<student code="p3960134">` is a `student` element with the attribute `code` having the value `p3960134`.
- **Entities** are used to represent special characters, such as '&', '<', '>', and also to refer to often repeated or varying text and to include the content of external files. Every entity must have a unique name. Entity references begin with the ampersand (&) and end with a semicolon (;). For example, to use a literal '<' into a document we should refer to it with `&lt.`
- **Comments** begin with `<!--` and end with `-->`. Comments are not part of the textual content of an XML document. An XML processor is not required to pass them along to an application. For example:
  `<!-- Example of a simple XML file -->`
- **Processing instructions** (**PIs**) are a mechanism to provide information to an application. Like comments, they are not textually part of the XML document, but the XML processor is required to pass them to an application. Processing instructions have the form `<?name pidata?>`. The name, called the PI target, identifies the PI to the application. Applications should process only the targets they recognize and ignore all other PIs. Any data that follows the PI target is optional and is intended for the application that recognizes the target. PI names beginning with `xml` are reserved for XML standardization, like the *XML declaration* `<?xml version="1.0">` in the beginning of the previous document, which identifies the document as an XML document and indicates the version of XML to which it was authored.
- **Marked sections** or **`CDATA` section**s instruct the parser to ignore most mark-up characters. Between the start of the section, `<![CDATA[`, and the end of the section, `]]>`, all character data is passed directly to the application, without interpretation. Elements, entity references, comments, and processing instructions are all unrecognised and the characters that comprise them are passed literally to the application. This is useful in cases of source code listing in an XML document that might contain characters that the XML parser would ordinarily recognize as mark-up. For example:

```
<![CDATA[
*p=&q; a = ( c != b);
]]>
```

The XML Specification explicitly says XML uses **ISO 10646**, the international standard 31-bit character repertoire which covers most human (and some non-human) languages. This is currently congruent with Unicode. The spec says: 'All XML processors must accept the UTF-8 and UTF-16 encodings of ISO 10646...Regardless of the specific encoding used, any character in the ISO 10646 character set may be referred to by the decimal (`&#dddd`) or hexadecimal (`&#xHHHH`) equivalent of its bit string'.

2.4 DTDs in XML

2.4.1 Introduction in DTDs

DTD stands for Document Type Declaration. Declarations allow a document to communicate meta-information to the parser about its content. Meta-information includes the allowed sequence and nesting of tags, attribute values and their types and defaults, the names of external files that may be referenced and whether or not they contain XML, the formats of some external (non-XML) data that may be referenced, and the entities that may be encountered.

XML does not require a document type declaration. However, a document type declaration should be supplied so as the XML document to be understood unambiguously. Most authoring environments need to read and process document type declarations in order to understand and enforce the content models of the document. Also, if an XML document relies on default attribute values (see later in this chapter), at least part of the declaration must be processed in order to obtain the correct default values. In applications where a person composes or edits the data (as opposed to data that may be generated directly from a database, for example), a DTD is probably going to be required if any structure is to be guaranteed.

The document type declaration (DTD), if it is present, must be the first thing in the document after optional processing instructions and comments. The DTD identifies the root element of the document and may contain additional declarations. All XML documents must have a single root element that contains all of the content of the document. Additional declarations may come from an external DTD, called the *external subset*, or be included directly in the document, the *internal subset*, or both.

Declarations in the internal subset override declarations in the external subset. The XML processor reads the internal subset before the external subset and the *first* declaration takes precedence. The *standalone document declaration*, which occurs in the XML declaration in the beginning of an XML document, may have a value of `yes` that indicates that only internal declarations need to be processed while a value of `no` indicates that *both* the internal and external declarations must be processed. [WALS98]

Since the construction of a DTD for a given DTD-less XML file is a difficult and time-consuming task, there has been designed a tool to generate XML DTDs by itself. SAXON DTDGenerator is a program that takes an XML document as input and produces a Document Type Definition (DTD) as output to which the document would conform. It could be found at the following URL:

http://users.iclway.co.uk/mhkay/saxon/dtdgen.html

DTDGenerator Front-end is also a Perl script written by Paul Tchistopolskii. Users may upload their files at this URL: http://www.pault.com/Xmltube/dtdgen.html and generate a DTD on the fly.

2.4.2 Syntax of DTDs in XML

The Document Type Declaration for the XML file of the example of the previous chapter can be the following one:

```
<!DOCTYPE example [
<!ELEMENT example (student)*>
<!ELEMENT student (name, department, address?)>
<!ATTLIST student code ID #REQUIRED>
<!ELEMENT name (firstname, lastname)>
```

```
<!ELEMENT firstname (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT department (#PCDATA)>
<!ATTLIST department year CDATA>
<!ELEMENT address (street, city)>
<!ELEMENT street (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ATTLIST city zipcode CDATA> ]>
```

This DTD identifies the element <example> as root element of the XML document, which contains all of the content of the document. The name specified in the DOCTYPE statement must be the same as the name of the root element. In this case, the DTD is internal, since the standalone document declaration of the original XML document from the previous paragraph, had the value yes (standalone="yes"). If the DTD is external, the DOCTYPE statement still occurs in the document, with the argument "SYSTEM -filename-", where "-filename-" is the name of the file containing the DTD. For example, if the above DTD were in an external file called "xxx.dtd", the DOCTYPE statement would read:

<!DOCTYPE example SYSTEM xxx.dtd>

The same line would then also appear as the first line in the file xxx.dtd.

There are four kinds of declarations in XML: element type declarations, attribute list declarations, entity declarations, and notation declarations, all of which begin with <!.[WALS98][REC98]

- **Element type declarations** identify the names of elements and the nature of their content, usually referred as *content model*. The content model defines what an element may contain. For example:

  ```
  <!ELEMENT student (name, department, address?)>
  ```

  This declaration identifies the element named student, which must contain name and department and may contain address. The commas between element names indicate that they must occur in succession. A plus '+' after a name indicates that it may be repeated more than once but must occur at least once. A question mark '?' after a name, like address, indicates that it is optional (it may occur zero or exactly one time). A name with no punctuation, like name, must occur exactly once. An asterisk '*' after the content model, as for the element example, indicates that the content is optional (may occur zero or more times).

  In addition to element names, the special symbol #PCDATA is reserved to indicate character data. The moniker PCDATA stands for parseable character data. Elements that contain only other elements are said to have *element content*. Elements that contain both other elements and #PCDATA are said to have *mixed content*. Two other content models are possible: EMPTY indicates that the element has no content (and consequently no end-tag), and ANY indicates that *any* content is allowed.

- **Attribute list declarations** identify which elements may have attributes, what attributes they may have, what values the attributes may hold, and what value is the default. Each attribute in a declaration has three parts: a name, a type, and a default value. An attribute list declaration from our example:

  ```
  <!ATTLIST city zipcode CDATA>
  ```

The `city` element has an attribute named `zipcode`, which is type of string (character data), has not default value and is not required.

There are six possible **attribute types**:

- `CDATA(Character DATA):` Strings with any text allowed;
- `ID:` IDs uniquely identify individual elements in a document. The value of an ID attribute must be a name unique in the document. Elements can have only a single ID attribute;
- `IDREF:` The value of a single ID attribute on some element in the document;
- `ENTITY:` The name of a single entity;
- `NMTOKEN:` Name token attributes are a restricted form of string attribute; and
- `List of names(enumerated type):` The value of the attribute must be taken from a specific list of names. Each of the possible values is explicitly enumerated in the declaration.

There are four possible **default values**:

- `#REQUIRED:` Attribute must have an explicitly specified value on every occurrence of the element;
- `#IMPLIED:` Attribute value is not required, and no default value is provided;
- "value" : Attribute value is not required on each element in the document, and if it is not present, it will appear to be the specified default, which can be any legal value; and
- `#FIXED` "value": Attribute is not required, but if it occurs, it must have the specified value. If it is not present, it will appear to be the specified default.

- **Entity declarations** associate a name with some other fragment of content, which can be a piece of regular text, a piece of the document type declaration, or a reference to an external file containing either text or binary data. There are three kinds of entities:
  - *Internal entities* associate a name with a string of literal text, allowing to define shortcuts for frequently typed text or text that is expected to change. An internal entity declaration is like:

    ```
    <! ENTITY AUEB "Athens University of Economics and Business">
    ```

    Using `&AUEB;` anywhere in the document will insert 'Athens University of Economics and Business' at that location. The XML specification predefines five internal entities: `<` for `<`, `>` for `>`, `&` for `&`, `&apos;` for `'`, and `&quot;` for `"`.
  - *External entities* associate a name with the content of another file allowing an XML document to refer to its contents. External entities contain either text or binary data. If they contain text, the content of the external file is inserted at the point of reference and parsed as part of the referring document. Binary data is not parsed and may only be referenced in an attribute. Binary data is used to reference figures and other non-XML content in the document. An example of the first case with text:

    ```
    <!ENTITY outfile SYSTEM "/standard/exam.xml">
    ```

    Using `&outfile;` will insert the *contents* of the file `/standard/exam.xml` at the location of the entity reference.
  - *Parameter entities* can only occur in the document type declaration. A parameter entity declaration is identified by placing % in front of its name and is also referenced in the same way. Parameter entity references are immediately expanded in the document type declaration and their replacement text is

part of the declaration. Parameter entities are not recognized in the body of a document. The advantage of using a parameter entity is two-fold. First, it allows giving a descriptive name to the content, and second it allows changing the content model in only a single place. For example:

```
<!ENTITY % name "#PCDATA">
```

And then we can declare `firstname` and `lastname` as following:

```
<!ELEMENT firstname (%name;)>
<!ELEMENT lastname  (%name;)>
```

- **Notation declarations** identify specific types of external binary data. This information is passed to the processing application. A typical notation declaration is:

```
<!NOTATION GIF87A SYSTEM "GIF">
```

2.4.3 Validity

Well-formed Documents

- A document can only be **well-formed** if it obeys the syntax of XML. A document that includes sequences of mark-up characters that cannot be parsed or are invalid cannot be well-formed. By definition, if a document is not well-formed, it is not XML.
- If there is no DTD in use, the document must start with a Standalone Document Declaration saying so.
- All tags must be balanced, which means that all elements that may contain character data must have both start- and end-tags present.
- All attribute values must be in quotes. The single-quote character may be used if the value contains a double-quote character, and vice versa.
- Any `EMPTY` element tags must either end with '`/>`' or appear non-`EMPTY` with the addition of a real end-tag.
- There must not be any isolated markup-start characters (`<` or `&`) in the text data (i.e. they must be given as `<` and `&`).
- Elements must nest inside each other properly, that is no overlapping markup is allowed.
- Well-formed files with no DTD may use attributes on any element, but the attributes must all be of type CDATA by default.

Valid Documents

A well-formed document is **valid** only if it contains a proper document type declaration and if the document obeys the constraints of that declaration (element sequence and nesting is valid, required attributes are provided, attribute values are of the correct type, etc.). An XML version of the specified DTD must be accessible to the XML processor, either by being available locally (*i.e.* the user already has a copy on disk), or by being retrievable via the network.

2.4.4 Namespaces

An XML namespace is a collection of names that can be used as elements or attribute names in an XML document. The namespace qualifies element names uniquely on the Web in order to avoid conflicts between elements with the same name. The namespace is identified by some URI (Universal Resource Identifier), either a URL (Uniform Resource Locator) or a URN (Uniform Resource Number). Namespaces can be declared either explicitly or by default. With an explicit declaration a shorthand, or prefix, is defined to substitute for the full name of namespace, qualifying thus elements belonging to that namespace. Explicit declarations are used when a node contains elements from different namespaces. A default declaration declares a namespace to be used for all elements within its scope, and a prefix isn't used. The attribute "xmlns" is the XML keyword used for a namespace declaration. [MICR]

The following explicit declaration declares "bk" and "money" to be shorthand for the full names of their respective namespaces:

```
<bk:BOOK xmlns:bk="urn:BookLovers.org:BookInfo" xmlns:money="urn:Finance:Money">
```

All elements, afterwards, beginning with `"bk:"` or `"money:"` are considered to be from the namespace "urn:BookLovers.org:BookInfo" or "urn:Finance:Money," respectively.

2.4.5 Schemas

An XML Schema is an XML-based syntax, or schema, for defining how an XML document is marked up. XML Schema is a schema specification recommended by Microsoft. [MICR] It improves on DTDs in several ways, including the use of XML syntax and support for data-typing and namespaces. An XML Schema allows specifying an element as an integer, a float, a boolean, a URL etc, while a DTD allows defining an element only as a string.

A Schema begins with the Schema element containing the declaration of the schema namespace and the declaration of the data types namespace, if there are any. The declaration elements used to define elements and attributes are the following:

- ElementType: Assigns a type and conditions to an element, and what, if any, child elements it can contain;
- AttributeType: Assigns a type and conditions to an attribute;
- attribute: Declares that a previously defined attribute type can appear within the scope of the named ElementType element; and
- element: Declares that a previously defined element type can appear within the scope of the named ElementType element.

The content of the schema begins with the AttributeType and ElementType declarations of the innermost elements. When an element has attributes or child elements they must be previously declared in their own ElementType or AttributeType declaration. ElementType and Attribute Type declarations must precede attribute and element content declarations that refer to these types. The XML Schema information can reside either in a schema file or within the XML document itself.

2.4.6 Data types

A data type within an XML document is a type that has been assigned to an element on the instance using the dt:dt attribute or through an XML Schema. In addition, data types can be declared as elements. The XML parser uses the data type information to validate the document.

The declaration `'xmlns:dt="urn:schemas-microsoft-com:datatypes"'` assigns `dt` to the namespace "urn:schemas-microsoft-com:datatypes". Any type qualified with dt will be from that namespace. The schema and data type namespaces are declared at the beginning of the XML Schema so that the prefix dt can be used to show which type attributes hold data type designations.

```
<Schema xmlns="urn:schemas-microsoft-com:xml-data" xmlns:dt="urn:schemas-
microsoft-com:datatypes">
<ElementType name="NUMBER" content="textOnly" dt:type="number"/>
```

2.5 XML APIs

There are two major types of XML APIs (Application Program Interfaces):

- Tree-based APIs; and
- Event-based APIs.

They were both created to give access to the information stored in XML documents using any programming language (e.g. Java, C++, Perl, Python, etc.) and any parser for that language.

2.5.1 Tree-based APIs - DOM

A **tree-based API** compiles an XML document into an internal tree structure, then allows an application to navigate that tree.

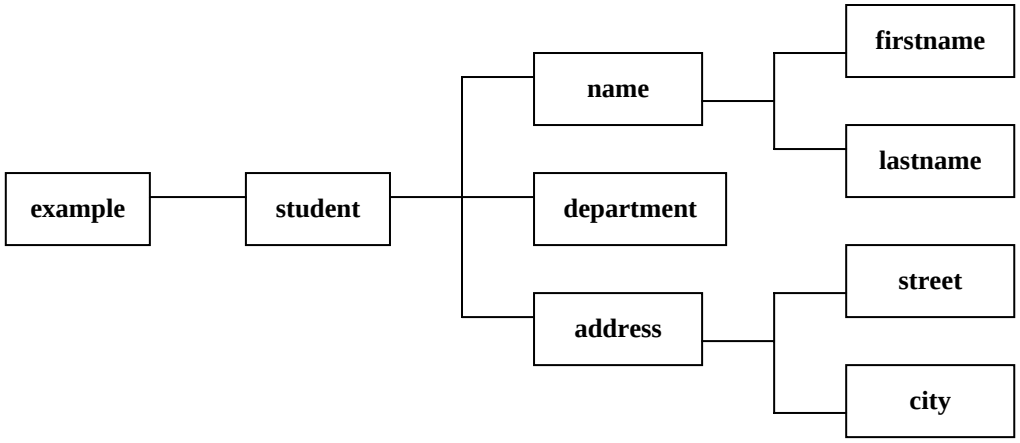
Document Object Model (**DOM**) is a tree-based API. DOM Level 1 (1998) is a platform and language neutral interface that allows programs and scripts to dynamically access and update the content, structure and style of documents. DOM provides a standard set of objects for representing any well-formed HTML and XML documents, a standard model of how these objects can be combined, and a standard interface for accessing and manipulating them. Vendors can support the DOM as an interface to their proprietary data structures and APIs, and content authors can write to the standard DOM interfaces rather than product-specific APIs, thus increasing interoperability on the Web.

The DOM Level 2 (2000) builds on the DOM Level 1. DOM Level 2 adds interfaces for a Cascading Style Sheets object model, an event model, and a query interface, amongst others. DOM Level 2 is made of a set of core interfaces to create and manipulate the structure and contents of a document and a set of optional modules. These modules contain specialized interfaces dedicated to XML, HTML, an abstract view, generic stylesheets, Cascading Style Sheets, Events, traversing the document structure, and a Range object. The DOM Core represents the functionality used for XML documents, and also serves as the basis for DOM HTML. The Level 1 interfaces were extended to provide both Level 1 and Level 2 functionality

DOM, in general, gives access to the information stored in an XML document as a *hierarchical* object model. DOM creates a tree of nodes based on the structure and information in the XML document regardless of the kind of information (whether it is tabular data, or a list of items, or just a document). DOM preserves the sequence of the elements that it reads from XML documents, because it treats everything as it if were a document. In DOM, the parser does almost everything, read the XML document in, create a Java object model on top of it and then give a reference to this object model (a Document object) so that the user can manipulate it. [IDRI99]

The XML object model is a collection of objects used to access and manipulate the data stored in an XML document. The XML document is modeled like a tree, in which each element in the tree is considered a node. Each node contains the actual data in the document. The root, or document element, is the top-level node from which its child nodes branch out to form the XML tree. The root node can only appear in the document once.

The tree model for the example XML document of this chapter is the following:



Objects with various properties and methods represent the tree and its nodes. A sample of the available properties and methods is illustrated in the following table:

Property / Method	**Returns**
XMLDocument	Reference to the XML Document Object Model (DOM)
DocumentElement	Document root
ChildNodes	Node list containing the children of a node
Item	Zero-based index for accessing individual nodes within the list
Text	Text content of the node

For example, XMLDocument.documentElement.childNodes.item(1).text returns the text content of the second child of the root node. [MICR]

The following figure is a schematic of a general system that can transform one XML document to some other form programmatically. We use a DOM parser to parse an XML file, and the parser returns a tree that is an exact representation of the XML in the file. We can then traverse the document tree in software and add nodes, delete them, update their values, read or set their attributes. When the tree has the new structure we desire, we can print the top-self node to another XML file, and the new document is created. [JOHN00]

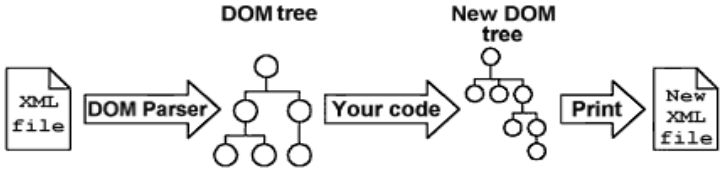


Figure: A DOM document transformation system

DOM is an ideal solution when the XML documents contain document data, for example in a document information management system. However, if documents contain mostly structured data DOM is not the best choice. [IDRI99]

2.5.2 Event-based APIs - SAX

An **event-based API**, on the other hand, reports parsing events, such as the start and end of elements, directly to the application through call-backs, and does not usually build an internal tree. The application implements handlers to deal with the different events much like handling events in a graphical user interface.

Tree-based APIs are useful for a wide range of applications, but they often put a great strain on system resources, especially if the document is large. Furthermore, some applications need to build their own, different data trees, and it is very inefficient to build a tree of parse nodes, only to map it onto a new tree.

In both of these cases, an event-based API provides a simpler, lower-level access to an XML document: users can parse documents much larger than the available system memory, and they can construct their own data structures using their callback event handlers. An event-based API is ideal for pattern searching.

For example, if we had to locate the record element containing the word "Athens" and the XML document was 20MB large (or even just 2MB), it would be very inefficient to construct and traverse an in-memory parse tree, like in the DOM case, just to locate this one piece of contextual information; an event-based interface would allow to find it in a single pass using very little memory.

To understand how an event-based API can work, we could consider the following sample document:

```
<?xml version="1.0">
<doc>
<par>Hello, world!</par>
</doc>
```

An event-based interface will break the structure of this document down into a series of linear events:

```
start document
start element: doc
start element: par
characters: Hello, world!
end element: par
end element: doc
end document
```

An application handles these events just as it would handle events from a graphical user interface: there is no need to cache the entire document in memory or secondary storage.

Finally, it is possible to construct a parse tree using an event-based API, and it is possible to use an event-based API to traverse an in-memory tree. [MEGG00]

SAX (Simple API for XML) is a standard interface for event-based XML parsing, developed collaboratively by the members of the XML-DEV mailing list. SAX 1.0 was released on May 1998, and is free for both commercial and non-commercial use. SAX implementations are currently available in Java and Python. The final version of SAX2/Java was scheduled for release on May 2000 [MEGG00]. SAX gives access to the information in the XML document, not as a tree of nodes, but as a sequence of *events*. SAX does not create a default Java object model on top of the XML document like DOM does. This makes SAX faster but also necessitates from users the following tasks:

1. Creation of their own custom object model to hold the information in the XML document; and
2. Creation of a XML *document handler* class that listens to SAX events, interprets them and properly creates the object model.

All SAX requires is that the parser should read in the XML document, and fire a bunch of events depending on what tags it encounters in the XML document. SAX will fire an event for every open tag, and every close tag. It also fires events for #PCDATA and CDATA sections, for processing instructions, DTDs, comments, etc. The document handler will have to interpret these events and the sequence in which these events are fired. The SAX document handler does element to object mapping.

If the information stored in the XML documents is machine readable and generated data (such as queries that are formulated using some kind of text based query language (SQL, XQL) or result sets that are generated based on queries) then SAX is the right API for giving access to this information. [IDRI99]

2.6 Navigation with XML

2.6.1 Introduction in Links

A link expresses a relationship between resources. A resource is any location (an element, or its content, or some part of its content, for example) that is addressed in a link. The exact nature of the relationship between resources depends on both the application that processes the link and semantic information supplied.

The XML Linking Language (**XLL**), based on SGML, HyTime and the Text Encoding Initiative (TEI), introduces a standard linking model for XML. XLL consists of two components, **X-Link** and **X-Pointer** (see 2.6.2 Types of XML Links). XLL enables:

- Bi-directional linking;
- Filtered views of data;
- Persistent links that have semantics attached to them (attributes on links);
- Dynamic document assembly;
- Publish dynamic updates or software patches;
- Support annotations; and
- Manage links so that they don't break.

Since XML does not have a fixed set of elements, the name of the linking element cannot be used to locate links. Instead, XML processors identify links by recognizing the `xml:link` attribute. Other attributes can be used to provide additional information to the XML processor, allowing for some control over the linking

behaviour. The `show` attribute determines whether the linked document is embedded in the current document, replaces the current document or is displayed in a new window when the link is traversed. The `actuate` attribute determines how the link is traversed, either automatically or when selected by the user.[MALE98],[WALS98]

2.6.2 Types of XML links

- A **Simple Link** strongly resembles an HTML <A> link. It identifies a link between two resources, one of which is the content of the linking element itself. This is an in-line link :

```
<link xml:link="simple" href="locator">Link Text</link>
```

The *locator* identifies the other resource. The locator may be a URL, a query, or an Extended Pointer (see below).

- X-Link introduces **Extended Links**. Extended Links can involve more than two resources. Extended Links allow to express relationships between more than two resources:

```
<elink xml:link="extended" role="annotation">
<locator xml:link="locator" href="text.loc">The Text</locator>
<locator xml:link="locator" href="annot.loc">Annotations</locator>
<locator xml:link="locator" href="litcrit.loc">Literary Criticism</locator>
</elink>
```

This example shows how the relationships between a literary work, annotations, and literary criticism of that work might be expressed. Extended Links can be in-line, so that the content of the linking element (other than the locator elements) participates in the link as a resource, or ,like the example above, out-of-line links, which do not use their content as a resource.

- X-Pointer introduces **Extended Pointers.** X-Pointer adds advanced addressing into the XML document structure. X-Pointer is a language for pointing into the sub-parts of a document. In particular, X-Pointers allow to locate arbitrary resources in a document, without requiring that the resource be identified with an ID attribute. Location pointers are a series of steps to define a path to a particular place in a document. X-Pointers offer a syntax that allows to locate a resource by traversing the element tree of the document containing the resource. For example :

```
child(2,student).(3,.)
```

locates the third child (whatever it may be) of the second `student` in the document of our example.

In addition to selecting by elements, X-Pointers allow for selection by ID, attribute value, and string matching. X-Pointers are particularly useful when the document being linked-to has not defined anchors where links are desired and/or it is not possible to modify the document to which we wish to link. [WALS98]

2.7 Usage of XML

2.7.1 Applications

There is a large range of applications that cannot be implemented efficiently and effectively within the limitations of previous technologies like HTML, and could only be accomplished with the powerful features and possibilities XML offers. These applications can be divided into four broad categories: [BOSA97]

1. Applications that require the Web client to mediate between two or more heterogeneous databases.

   A way to enable interchange between heterogeneous systems is to adopt a single industry-wide interchange format that serves as the single output format for all exporting systems and the single input format for all importing systems.

   A number of industries, including the aerospace, automotive, telecommunications, and computer software industries, have been using hub languages to perform data interchange for years. Typically, the major players in an industry form a standards consortium tasked with defining a Document Type Definition, which is the way in which the tag set and grammar of a mark-up language are defined. This DTD can then be sent with documents that have been marked up in the industry standard language using off-the-shelf editing tools, and any standard application on the receiving end can validate and process them.

   The XML solution is system-independent, vendor-independent, and proven by over a decade of SGML implementation experience. XML merely extends this proven approach to document interchange over the Web.

2. Applications that attempt to distribute a significant proportion of the processing load from the Web server to the Web client. The advantages of using XML in distributed processing are the following:

   - Industry-specific mark-up cannot be implemented within the confines of the fixed HTML tag set.
   - The data representation must be platform and vendor independent so that data from a variety of sources can be used to drive a variety of distributed applications (some of which may be provided by third parties, generating a sub industry of providers of tools that can work with the standardized data stream).
   - A computation-intensive process that would otherwise entail an enormous, extended resource hit on the server has been changed into a brief interaction with the server followed by an extended interaction with the user's own Web client.

3. Applications that require the Web client to present different views of the same data to different users, without requiring that the data be downloaded again in a different form from the Web server.

   One application in this category is dynamic tables of contents. It is possible using Web servers built on object-oriented databases to present the user with a table of contents into a large collection of data that can be expanded with a mouse click to "open up" a portion of the TOC and reveal more detailed levels of the document structure. Dynamic TOCs of this kind can be generated at run time directly from the hierarchical structure of the document. Unfortunately, the Web latency built into every expansion or contraction of the TOC makes this process sluggish in many user environments. A much better solution is to download the entire structured TOC to the client rather than just individual server-generated views of the TOC. Then the user can expand, contract, and move about in the TOC supported by a much faster process running directly on the client. Instead, standard XML editors may be used to create structured content from which a structured TOC could be generated at run time and downloaded to browsers that would automatically create and display the TOC using either a downloaded Java applet or a standard set of JavaHelp class libraries.

The ability to capture and transmit semantic and structural data made possible by XML greatly expands the range of possibilities for client-side manipulation of the way data appears to the user. For example:

- An installation sheet that carries warnings in multiple languages can be made to show just the ones in the language selected by the user.
- A document containing many annotations can be switched from a mode that shows only the text, to a mode that shows only the annotations, to a mode that shows both, just by making a menu selection.
- A phone book sorted by last name can instantly be changed into a phone book sorted by first name.

XML creates the potential for variances among hardware, software, and human readers of the data. The structure of XML allows for the deconstruction of Web pages into parts that can be sent to any kind of network-attached device. Software also has an ability to view XML in these discrete pieces. Finally, humans also gain the benefit of adaptable views. With multiple style mechanisms, it is possible to create many views of the same data. [MIKO98]

4. Applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users (See 2.7.4 XML Agents).

2.7.2 Browsers

Since XML is a relatively new language in the computers community, there is still poor support in browsers. The W3C hasn't issued any guidelines on what an "XML browser" would look like, or much suggestion as to how to process XML and HTML when the two are mixed together in a single document. In fact there are not exclusive XML-capable browsers yet, but some HTML-browsers can present XML data quite efficiently. The first browser to support XML was MS Internet Explorer 4.

Microsoft® **Internet Explorer** browser for Windows (5.5 preview release) and Macintosh (5.0) renders XML content best relying on eXtensible Style sheet Language Transformations (XSLT, see Chapter 3 about XSL) and provides only a minimal level of Cascading Style Sheets (CSS, see Chapter 3) support for use with XML. Microsoft appears to be remaining dependent upon HTML for the presentation and linking tasks. The XML parser in Internet Explorer 5 can validate an XML document with both a DTD and an XML Schema. Microsoft is also the architect of a hybrid solution (data islands, see Appendix A) in which fragments of XML can be embedded in HTML files, because current HTML-only browsers simply ignore element markup that they don't recognize.

MS Internet Explorer 5.x (http://www.microsoft.com/windows/ie/default.htm) ships with a C++ Data Source Object (DSO) that can be used to bind XML to HTML. Microsoft® Internet Explorer 4 shipped with a Java XML DSO that is still supported in Internet Explorer 5; however, the new C++ DSO gives better performance and the ability to bind directly to an XML data island. The C++ DSO provides users with the ability to create XML-driven Web applications in a completely declarative fashion, although it is still possible to write scripts against the XML document object. With the C++ DSO, both the ActiveX® Data Object (ADO) and XML object models are available to users. [LAUR00][MICRO]

Mozilla is an open-source web browser, designed for standards compliance, performance and portability. Netscape is a major contributor to the Mozilla code. At the same time, Netscape uses the Mozilla code as the basis of its Netscape 6 product. Hence, the work for the publicly released Netscape code for version 6 has been generally known as Mozilla project (http://www.mozilla.org). The XML support in Mozilla is built on James Clark's non-validating *expat* XML parser (see 2.7.3 Parsers). The output from *expat* is fed into a DOM-tree builder and document structures can be styled with CSS just like HTML documents. Users can

navigate and print XML documents just like they do HTML documents. Mozilla ignores external parsed entities and entities declared outside of the actual document. If entities are to be used in Mozilla, they need to be declared in the internal subset. Entity references don't appear on screen as part of the content - they just disappear. Mozilla provides a bin/dtd folder where additional entities can be stored. This is used to provide support for things XML formats like MathML, but isn't a readily available option for most developers. Mozilla also supports namespaces. Namespace support will also be critical for XLink when support arrives for more recent XLink drafts. The solid XML+CSS core and the underlying DOM support suggests that Mozilla will be a useful platform for building applications, not just web pages.

Opera 4.0 beta2 browser (http://www.opera.no/) renders XML content using an overlapping, but somewhat different, set of tools from those used by Mozilla. While much of the core is the same, some of the details, especially linking, are different. Opera 4.0 is built on the same combination of XML and Cascading Style Sheets (CSS) that the Mozilla project has used as its base. [LAUR00]

DocZilla is a multi-everything browser, which reads HTML, XML, and SGML, with XSL and CSS style sheets, runs under NT and Linux and is currently in Alpha version (http://www.doczilla.com). It is by far the most ambitious, and the only one so far backed by solid SGML expertise. It is a product of the cooperation between the authors of the *MultiDoc Pro* SGML browser, CITEC, and Mozilla. [UCC99]

In Appendix B, a special chart displaying the various levels of XML support in the last-generation browsers presented here is illustrated.

2.7.3 XML Parsers

A software module called an **XML processor** is used to read XML documents and provide access to their content and structure on behalf of another module, called the **application.** Conforming XML processors fall into two classes: validating and non-validating. Validating and non-validating processors alike must report violations of well-formedness constraints in the content of the document entity and any other parsed entities that they read.

Validating processors must report violations of the constraints expressed by the declarations in the DTD, and failures to fulfil the validity constraints. To accomplish this, validating XML processors must read and process the entire DTD and all external parsed entities referenced in the document.

Non-validating processors are required to check only the document entity, including the entire internal DTD subset, for well-formedness. While they are not required to check the document for validity, they are required to **process** all the declarations they read in the internal DTD subset and in any parameter entity that they read, up to the first reference to a parameter entity that they do *not* read. They must use the information in those declarations to normalize attribute values, include the replacement text of internal entities, and supply default attribute values. They must not process entity declarations or attribute-list declarations encountered after a reference to a parameter entity that is not read, since the entity may have contained overriding declarations.

The behaviour of a validating XML processor is highly predictable; it must read every piece of a document and report all well-formedness and validity violations. Less is required of a non-validating processor; it need not read any part of the document other than the document entity. This has two effects that may be important to users of XML processors:

- Certain well-formedness errors, specifically those that require reading external entities, may not be detected by a non-validating processor; and
- The information passed from the processor to the application may vary, depending on whether the processor reads parameter and external entities.

For maximum reliability in interoperating between different XML processors, applications that use non-validating processors should not rely on any behaviour not required of such processors. Applications that require facilities such as the use of default attributes or internal entities that are declared in external entities should use validating XML processors.

Lark and **Larval** are two XML processors written by **Tim Bray.** Lark is a non-validating XML processor implemented in the Java language, which attempts to achieve good trade-offs among compactness, completeness, and performance. Larval is a validating XML processor built on the same code base as Lark. The versions in use were the final beta of version 1.0 of Lark, and release 0.8 of Larval (January 1998). Lark is available on the Internet for general public use at the following URL: http://www.textuality.com/Lark/.

Lark currently has no user interface at all. Lark is a processor only and does not attempt to validate. It does read the DTD, with parameter entity processing, it processes attribute list declarations to find default values and entity declarations. Lark is relatively full-featured; it implements everything in the XML spec and reports violations of well-formedness. Lark's error-handling is draconian. After encountering the first well-formedness error, no further internal data structures are built or returned to the application. However, Lark does continue processing the document looking for more syntax errors. Larval is a full validating XML processor and reports violations of validity constraints, but does not apply draconian error handling to them.

Lark adopts in fact an event-based API, following internally the same philosophy as SAX. Lark presents as a set of Java classes. From an application's point of view, the `Lark` and `Handler` classes are central. The `Lark` class recognizes significant objects in the XML document while the `Handler` class provides the application with the appropriate processing semantics. Along with presenting the event stream to the application, Lark can optionally build a parse tree, and if so doing, can optionally save copies of the XML document's character data, all in parallel with providing the event stream.

Expat (XML Parser Toolkit) is a library, written in C, for parsing XML documents. It's the underlying XML parser for the open source Mozilla project (Netscape 5), Perl's XML::Parser, and other open-source XML parsers. This library is the creation of James Clark, who was the technical lead on the XML Working Group at W3 that produced the XML specification. Expat is not a validating parser. It only reads the internal DTD subset. It doesn't process an external DTD nor parameter entity references. More information about expat can be found at the URL: http://www.jclark.com/Xml/General XML/expat.html while it can be downloaded via ftp at: ftp://ftp.jclark.com/pub/xml/expat.zip.

There is also an article on its use at: http://www.xml.com/pub/1999/09/expat/index.html

Microsoft® has also designed an **XML Validation Tool**. XMLINT.EXE is an updated version of the XMLINT command line tool that shipped in the IE4 SDK. This tool checks that a given XML file is well formed. It also uses the XML DOM to check that the document is also valid according to the DTD (or XML-Data Schema). The tool can be downloaded from the following URL, as well as a manual of the tool can be found: http://msdn.microsoft.com/xml/tools/xmlint.asp

Microsoft® XML Notepad is a simple prototyping application for HTML authors and developers that enables the rapid building and editing of small sets of XML-based data. With XML Notepad, developers can quickly create XML prototypes in an iterative fashion, using familiar metaphors. XML Notepad offers an intuitive and simple user interface that graphically represents the tree structure of XML data. Working with the standard building blocks of XML supported in Microsoft® Internet Explorer 4.0, authors are able to create reproducible data structures that can be easily filled, allowing greater emphasis to be placed on

application development instead of manual data structuring. The current version is beta 1.5 (released in May 1999) and can be downloaded from the following URL: http://msdn.microsoft.com/xml/notepad/intro.asp

2.7.4 XML Agents

An agent can be defined as a threaded object that collects information from more than one machine on a network on behalf of a user; in other words, an agent could be considered as a representative or emissary from a person. Agents are commonly categorized as "intelligent," "mobile," and "personal". XML Metadata allows an agent to be more personal in the sense that it has access to descriptions of the data that help it find what it is looking for. XML does not make an agent more mobile or more intelligent. [MIKO98]

By using XML, a very sophisticated personal news filter that spans multiple sites or the entire Internet could be created. The XML repository would provide the date stamp, enabling agents or search engines to filter the information to extract only the “new” information. Then, the information could be easily extracted, formatted, and delivered in any way the user chooses (such as an Active desktop, a personal news web page, email, pager) allowing all individuals to create “custom” newspapers with the latest information. [HOGA97]

A potential domain for XML applications would be related to intelligent Web agents demanding for structured data. Such applications could be those in which user preferences must be represented in a standard way to mass media providers.

For example a personalized TV guide that works across the entire spectrum of possible providers requires not only that the user's preferences and other characteristics (educational level, interest, profession, age, visual acuity) be specified in a standard, vendor-independent manner (in an industry-standard mark-up system) but also that the programs themselves be described in a way that allows agents to intelligently select the ones most likely to be of interest to the user. This second requirement can be met only by a standardized system that uses many specialized tags to convey specific attributes of a particular program offering (subject category, audience category, leading actors, length, date made, critical rating, specialized content, language, etc.). Exactly the same requirements would apply to customized newspapers and many other applications in which information selection is tailored to the individual user.

The implementation of these applications definitely require XML data in order to function in an interoperable way and thereby allow intelligent Web agents to compete effectively in an open market. [BOS97]

2.7.5 Metadata Formats

Since, as it is already mentioned, XML is really a meta-language, i.e. it can be used to describe other mark-up languages, several different metadata formats have been created in the XML language, each one satisfying the specific needs of a different field of applications, including:

- **RSS** (Rich Site Summary): A format for describing websites via “channels”. It is discussed in chapter 8 in a more detailed way.
- **RDF** (Resource Description Format): A powerful way to automatically describe what information is available. It is discussed in chapter 9 in a more detailed way.
- **CDF** (Channel Data Format): A format for publishing Web information on desktops. Designed for push technology.
- **PICS** (Platform for Internet Content Selection): A format initially intended to help create a "ratings" system on the Internet to protect children from adult-oriented material.

- **WIDL** (Web Interface Definition Language): An XML application proposed by webMethods Inc., which enables automation of all interactions with Web documents and forms. It consists of six XML-compliant HTML extenders that define a universal schema for HTML documents based on the Document Object Model (DOM). It provides a general method of representing request/response interactions over standard Web protocols and allows the Web to be utilized as a universal integration platform. It is a universal "API to the Web" – a meta-language that describes a service-based architecture over the document-based resources of the World Wide Web. It provides the structure necessary for generating client code in languages such as Java, C/C++, COBOL, and Visual Basic. WIDL enables a practical and cost-effective means for diverse systems to be rapidly integrated across corporate intranets, extranets, and the Internet. [SUSA97] [MIKO98]
- **OCS** (Open Content Syndication): The OCS Directory format is designed to enable channel listings to be constructed for use by portal sites, client based headline software and other similar applications.
- **Web Collections**: A meta-data syntax that fits easily within the framework of the World Wide Web. They can be expressed inside HTML documents or on their own. They are stylistically similar to HTML to enable easy authoring. Some of the anticipated applications of Web Collections include Web Maps, HTML Email threading, scheduling, content labelling, and distributed authoring.
- **MCF** (Meta Content Framework): An XML application proposed by Netscape Communications, which provides a standard way to describe files or collections of information.
- **XMI** (XML Metadata Interchange): A format that combines the benefits of the web-based XML standard for defining, validating, and sharing document formats on the web with the benefits of the object-oriented Unified Modelling Language (UML). It is a specification of the Object Management Group (OMG) that provides application developers a common language for specifying, visualizing, constructing, and documenting distributed objects and business models.
- **OSD** (Open Software Description): A format that provides an XML-based vocabulary for describing software packages and their inter-dependencies, whether it is user initiated ("pulled"), or automatic ("pushed"). OSD can be very useful in automated software distribution environments.
- **XLF** (Extensible Log Format): Its immediate goal is to design the XML-based web server log format. Its long-term goal remains to be the design of universal log format based on XML. XLF is XML-based Log Format designed to be extensible and universal.
- **WML** (The Wireless Application Protocol (WAP) Wireless Mark-up Language): A mark-up language based on XML intended for use in specifying content and user interface for narrowband devices, including cellular phones and pagers. A tag-based display language providing navigational support, data input, hyperlinks, text and image presentation, and forms. A browsing language similar to Internet HTML.
- **SDML** (Signed Document Mark-up Language): Its goal is to a) tag the individual text items making up a document, b) group the text items into document parts which can have business meaning and can be signed individually or together, c) allow document parts to be added and deleted without invalidating previous signatures, and d) allow signing, co-signing, endorsing, co-endorsing, and witnessing operations on documents and document parts.
- **SML** (SmartX Mark-up Language) is an implementation of XML for the smart card industry. The goal of SML is to enable automation of all interactions with XML documents providing general methods to represent a set of smart device functions. XML supports the creation of marker content that preserves data structure and promises web documents to be "machine-readable".
- **MathML** (Mathematical Mark-up Language): An XML application for describing mathematical notation and capturing both its structure and content. The goal of MathML is to enable mathematics

to be served, received, and processed on the Web, just as HTML has enabled this functionality for text.

- **CDIF** (CASE Data Interchange Format): A family of standards that lays out a single architecture for exchanging information between modelling tools, and between repositories, and defines the interfaces of the components to implement this architecture.
- **SMIL** (Synchronized Multimedia Integration Language): Allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can 1) describe the temporal behaviour of the presentation, 2) describe the layout of the presentation on a screen, and 3) associate hyperlinks with media objects. With SMIL, producing audio-visual content does not require learning a programming language and can be done using a simple text editor.
- **CML (**Chemical Mark-up Language): An XML application for the chemical community.

2.8 Future

XML has been heralded as "a true panacea" for all the Web's problems, as "the universal language" for data on the Web, as "a whole new processing paradigm", as "the second Web revolution" after the rise of Java, "a killer application" that is going to change everything from E-Commerce to object packaging and distribution, and several other metaphors.[KAM00]

XML has a very great significance if it is combined with Java, as Java and XML complement each other. Java provides platform-independence, while XML provides application-independence. XML provides a grammar that can be used to create self-describing data file formats. Thus, Java can be viewed as the universal Virtual Machine, and XML can be viewed as the universal Virtual Document. [MIKO98]

Information in an XML document is stored in plain text. Hence, it is easy to write parsers and all other XML enabling technology on different platforms. Moreover, by accepting and sending information in plain text format, programs running on disparate platforms can communicate with each other. Encoding information in plain text with tags is better than using proprietary and platform dependent binary formats. By taking the lowest common denominator approach, by being web enabled, protocol independent, network independent, platform independent and extensible, XML makes it possible for new systems and old systems, that are all different, to communicate with each other.

XML is an open standard; so making the W3C the keeper of the XML standard ensures that no vendor should be able to cause interoperability problems to occur between systems that use this open standard. By keeping all data in XML and using XML in communications protocols, companies can maximize the lifetime of their investment in their products and solutions. Language independence of XML also fosters immense interoperability amongst heterogeneous systems and is also good for future compatibility. By defining a set of programming language independent interfaces (DOM and SAX) that allows the accessing and mutation of XML documents, the W3C made it easier for programmers to deal with XML.

Since XML and HTML both derived from SGML, the current software and networking infrastructure available today to deal with HTML content can be re-used to work with XML. Even if clients don't support XML natively, Java with Servlets on the server side can convert XML with style sheets to generate plain HTML that can be displayed in all web browsers.

XML provides solutions for problems that have existed for the past 20 years. With most applications and software services using the Internet as a target platform for deployment, XML could not have come at a better time. As the Web becomes so popular, a new paradigm of computing has emerged, for which XML supplies one of the most important pieces, platform, vendor and application neutral data. Regardless of the programming language used to process XML, it will enable this new networked computing world. [IDRI99]

2.8.1 XML and e-business

What began as, and still is, a simple tagging language has emerged as a powerful electronic-business enabler - a mechanism for data interchange that is being infused into all levels of corporate infrastructures. Because XML on its own merely defines data, which means it must be properly formatted to provide any real-world benefit, questions have surfaced about how and where the technology should be implemented, and who - vendors, vertical industries, or individual companies - should dictate that decision.

XML will certainly be applied where structured information needs to be exchanged between heterogeneous systems or organizations for which a very tightly controlled standard won't work. One of the problems with XML, though, is that it's a low-level technology and can be applied to any number of problems, so people are still trying to figure out where it can be useful.

XML by itself is not a solution - it's a key ingredient to enable a richer type of solution. What companies need is an architecture where they can take XML data and integrate it with various applications. XML is clearly the language of choice for digital exchanges between companies within their respective industries, but the overarching question of who will define the XML standards, or vocabularies, remains, and XML's greatest strength - its simplicity - may also be its weak point. Another problem posed by the simplicity of XML is that the standard was designed to be highly extensible. That, combined with user confusion over how and where it should be implemented, has left the standard open to a variety of interpretations.

XML's greatest asset is not its extensibility - the feature that has been most widely leveraged to date - but its openness to transformation, which will allow different flavours of XML to talk to one another through adapters inside applications. So rather than vertically targeted document formats or vendor-specific implementations, transformation will make XML an integral component of business-to-business integration and interchange going forward. [LATT99]

According to S. Phipps, Chief of XML in IBM, “diversity in tagging vocabularies is inevitable, because businesses are so diverse. The most important XML standard is actually XSL, not XML. People are going to represent their business data structures as hierarchical data structures marked up with XML. Enterprises in the same business have a different business model. Therefore, it's likely that they will have different hierarchical data structures for the way they represent data. So, they will probably need to have slightly different vocabularies. Consequently, the lifeblood of all business-to-business in XML is going to be transformation. And if the lifeblood of XML is going to be transformation, it doesn't really matter if vendors produce slightly different XML standards because at least the data's all marked up. Enterprises are going to need strong transformation technologies in order to transfer from their version of whatever it is, to the version of another enterprise whatever it is. That's why the key to XML is not representation, but rather transformation.” [PHIP99]

Microsoft is expected to disclose a new strategy to develop Internet-based software and services called Next Generation Windows Services (NGWS). Microsoft and its competitors share the same "e-services" vision: In the future, people won't have to install software on their PCs or other Internet access devices. Instead, the software will be accessed through the Web as a service.

Core to the e-services strategy are XML, the Web standard for exchanging data, and new XML-based software tools that allow businesses with different computing systems to communicate with one another. Early examples of e-services include technologies that deliver stock quotes and weather reports to cell phones. Another example is renting software, such as accounting programs, over the Web through application service providers.

E-services will be also widely used in the business e-commerce market. A business can request a service over the Web and get responses back from companies that offer that service. A travel agency, for example, could send a request over the Web for cheap flights from San Francisco to London. Through XML technology, the travel agent could automatically receive information from airlines that fit search criteria.

PCs will remain the center of the computing universe but will be complemented by handheld devices, according to Gates. NGWS will be an operating system-like technology that will automatically update software, synchronize databases, and filter and compile information from multiple Web sites.

A key piece of the strategy is BizTalk Server 2000 (http://www.microsoft.com/biztalkserver/). BizTalk Server is Microsoft's XML-based software for linking computing systems and applications across the Net. A new feature within BizTalk Server, called "orchestration," allows businesses to easily define how e-commerce Web sites function and how information needs to be passed among mainframe, Unix, personal digital assistants, and Windows-based computers to complete a transaction. [WONG00]

2.8.2 XML - EDI

EDI (Electronic Data Interchange) works by providing a collection of standard message formats and element dictionary in a simple way for businesses to exchange data via any electronic messaging service.

The goal of XML/EDI is to deliver unambiguous and durable business transactions via electronic means, and also to establish a standard for commercial electronic data interchange that is open and accessible to all, and which delivers a broad spectrum of capabilities suitable to meet the full breadth of business needs. [GUID98]

XML/EDI provides a standard framework to exchange different types of data - for example, an invoice, healthcare claim, project status - so that the information which resides in a transaction, exchanged via an API, web automation, database portal, catalogue, a workflow document or message can be searched, decoded, manipulated, and displayed consistently and correctly by first implementing EDI dictionaries and then extending the vocabulary via on-line repositories to include the business language, rules and objects. Thus by combining XML and EDI a new powerful paradigm different from XML or EDI is created. [XML/EDI]

XML provides an ideal methodology for electronic business because:

- XML allows message type creators to clearly identify the role and syntax of each piece of interchanged data using a definition that is both machine processable and human interpretable;
- XML allows message type creators to identify the source of each shared structure using an Internet Uniform Resource Locator;
- XML allows message type creators to optionally identify which pieces of information should occur in each interchanged set of data and, where relevant, the order in which individual fields should occur in a particular message stream;
- XML documents can be given metadata fields that can be used to identify who is responsible for creating, transmitting, receiving and processing each message, and can have built-in facilities for identifying the storage points of programs that should be used to control processes; and
- XML can make use of facilities provided by the latest version of the Internet HyperText Transfer Protocol (HTTP), which can identify when a message should be moved from one stage of the interchange process to another, and to check that the relevant forms of interchange have taken place.

XML can be integrated with existing EDI systems by:

- providing application-specific forms that users can complete to generate EDI messages;
- generating EDI message formats for transmission between computers over the Internet, or through existing value-added networks (VANs); and

- allowing data received in EDI format to be interpreted according to sets of predefined rules for display by the receiver on standardized browsers using a user-defined template, rather than having to rely on specially customized display packages.

XML can extend existing EDI applications by:

- allowing message creators to add application-specific data to standardized message sets where required;
- allowing message creators or receivers to display the contents of each field in conjunction with explanatory material which is specific to the application and the language preferences of the user;
- allowing system developers to customize the help information associated with the data for each field; and
- allowing field value checking to be integrated with checks on the validity of the data with respect to information stored on local databases.[GUID98]

However, the integration of XML in all arenas of **E-Commerce** has several problems ranging from technical to social to political, eventually leading to a lack of standardization in several phases of business processes. Use of XML as a replacement of EDI warrants serious consideration for several reasons:

- EDI to XML transition may not always be cost-effective, with substantial investment already made in setting up the EDI system on a Value-Added Network (VAN);
- XML counterparts of EDI messages can be prohibitively large; and
- XML may not be a suitable choice due to weak datatyping in XML DTDs. Even if that situation is improved in the XML Schema effort, there is currently a lack of standardization where initiatives towards XML Schemas for E-Commerce have led to the state of chaos.

Use of XML in E-Commerce "gold rush" can be promising, as demonstrated by a variety of scenarios, but can lead to various pitfalls if lessons from the past failures are not learnt.[KAM00]

3. XSL

3.1 Introduction in XSL

XSL acronym stands for extensible style sheet language. XSL is a very powerful tool for transforming XML documents into other formats or a separate tree structure. Talking in a simple way, XSL is a style sheet language which was created in order to give us the opportunity to display XML documents the way (or to be exact, the format) we like. Currently, XSL transforms XML documents into a display format like the ones we see in Web browsers, however, in the near future XSL will be a complete viable XML presentation language.

XSL is being developed as a part of the W3C Style Sheets Activity (the first public working draft was published in August 18, 1998) and it is based on earlier specifications including CSS (Cascading Style Sheets) and DSSSL (Document Style Semantics and Specification Language).

To be more specific,"W3C continues to work with its Members, evolving the Cascading Style Sheets (CSS) language to provide even richer stylistic control, and to ensure consistency of implementations. W3C is also developing the Extensible Style sheet Language (XSL), which has document manipulation capabilities beyond styling."

According to the Activity description [July 1999], "XSL is a language quite different from CSS, and caters for different needs. The model used by XSL for rendering documents on the screen, builds upon many years of work on a complex ISO-standard style language called DSSSL. Aimed, by and large, at complex documentation projects, XSL has many uses associated with the automatic generation of tables of contents, indexes, reports and other more complex publishing tasks."

Capabilities provided by XSL as defined in the Proposal allow:

- formatting of source elements based on ancestry/descendency, position, and uniqueness;
- the creation of formatting constructs including generated text and graphics;
- the definition of reusable formatting macros;
- writing-direction independent style sheets; and
- extensible set of formatting objects.

In addition, XSL transformations address some common needs in XML. These include:

- **Enabling display:** The XSL transformation language enables the display of XML by transforming XML into grammar and structure suitable for display, for instance into HTML.
- **Direct browsing of XML files:** Internet Explorer 5 can apply XSL style sheets that produce HTML, allowing direct browsing of the XML files.
- **Content delivery to downlevel browsers:** XSL transformations can be executed on the server to provide HTML documents for downlevel browsers.
- **Schema translation:** The transformation process is independent of any particular output grammar and can be used for translating XML data from one schema to another.
- **Converting XML through querying, sorting and filtering:** The transformation can be used for general-purpose transformations within a single grammar, including filtering, sorting and summarizing data.

3.2 Design Principles

The following design principles have been used to guide the development of the XSL format, ordered from most important to least important:

- XSL should be straightforwardly usable over the Internet;
- XSL should be expressed in XML syntax;
- XSL should provide a declarative language to do all common formatting tasks;
- XSL should provide an “escape” into a scripting language to accommodate more sophisticated formatting tasks and to allow for extensibility and completeness;
- XSL will be a subset of DSSSL with the proposed amendment;
- A mechanical mapping of a CSS style sheet into an XSL style sheet should be possible;
- XSL should be informed by user experience with the FOSI style sheet language;
- The number of optional features in XSL should be kept to a minimum;
- XSL style sheets should be human-legible and reasonably clear;
- The XSL design should be prepared quickly;
- XSL style sheets shall be easy to create; and
- Terseness in XSL mark-up is of minimal importance.

3.3 Current Options For Displaying XML Documents

The options we have today for displaying XML documents are shown in the following diagram:

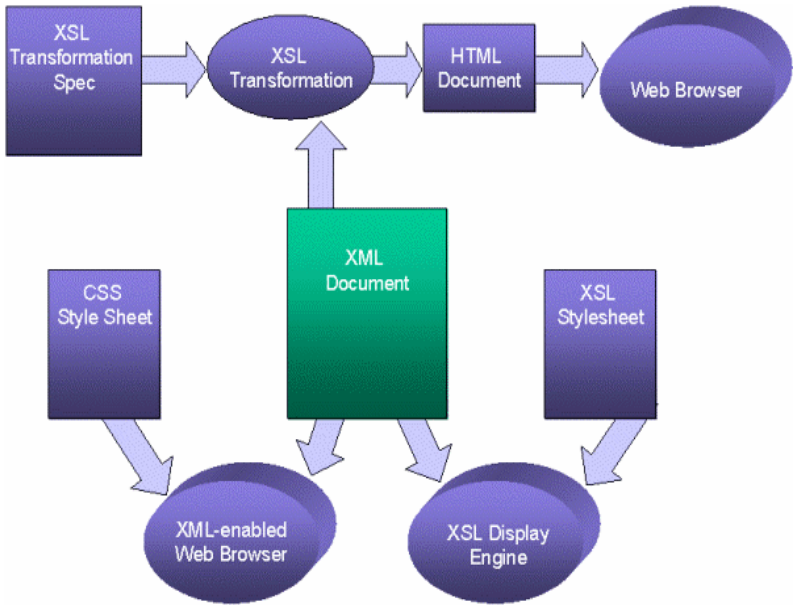


Figure: Options for Displaying XML documents

The presence of XSL is catalytic in order to transform the XML document into HTML format, which can be displayed in any of today's available web browsers. The transformation specifications are defining details about the presentation, which are considered by XSL transformation. The XML document can also be displayed in an XML-enabled web browser (such as Internet Explorer 5) using a CSS (Cascading Style Sheet), or in an XSL display engine which uses XSL style sheet. The last one is considered to be the most uncommon and less used method to display an XML document.

3.4 Why Use Style sheet

XSL is a style sheet language. A style sheet specifies the presentation of XML information using two basic categories of techniques:

- An optional transformation of the input document into another structure e.g. generation of constant text, sorting, moving text, duplicating text; and
- A description of how to present the transformed information.

In other words, a style sheet tells a processor how to convert logical structures (the source XML document represented as a tree) into a presentational structure (the result tree). We could say that an XSL style sheet is actually an XML document.

Style sheets are generally very useful for the following reasons:

- XML is not a fixed tag and has no application semantics;
- XML markup does not include formatting information;
- Multiple output formats;
- Reuse: the same content can look different in different contexts;
- Standardized styles: style sheets can be applied to the content at any time; and
- Freedom from style issues.

3.5 A Simple Description Of How XSL Works

We could say that an XSL processor 'works' like this:

First, it looks at an XML document and calls it source document. Then it examines the XML document and makes decisions based upon the way the XSL document is coded. It is possible, for example, to tell the XSL processor to render only the `text` element (not to be confused with a text node in general) in an XML document that may in fact be loaded with many more elements. The XSL processor can be told to do this in an XSL document by employing transformation mechanisms such as patterns. This technique may remind

somewhat of a mini-search engine. The process for developing patterns will look somewhat familiar to someone who has worked with SQL (the well known common database querying language).

The XSL processor searches for a pattern, and a series of one or more templates that match these patterns to return a result tree. The result tree is a third document instance that is derived from the application of the style sheet document to the source tree.

We could consider a situation in which only one small portion of an XML document needs to be transformed. Much of an XML document may be dedicated to information that is not necessary to show to anyone else. XSL allow us to simply choose which elements we want to display or send.

3.6 An Analytical Approach Of XSL Operation

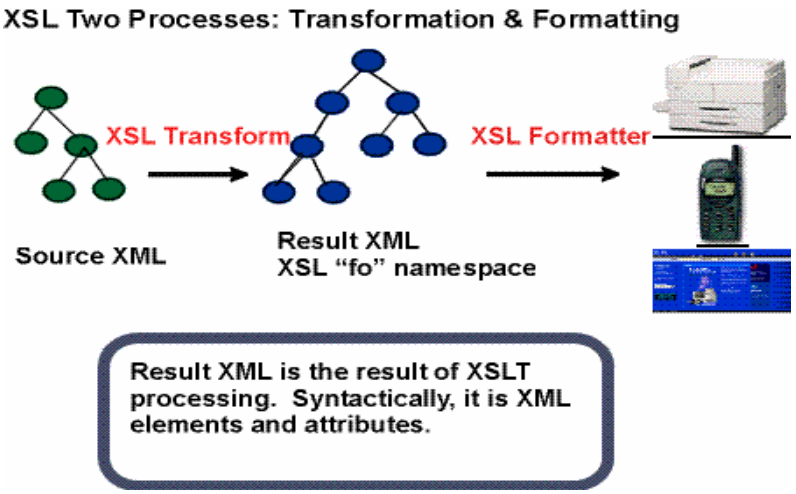
This chapter examines in depth the operation of XSL from the beginning, when the XML document is provided for transformation, until the termination of the procedure.

An XSL style sheet processor accepts a document or data in XML and an XSL style sheet and produces the presentation of that XML source content that was intended by the designer of that style sheet.

There are two aspects of this presentation process: first, constructing a result tree from the XML source tree and second, interpreting the result tree to produce formatted results suitable for presentation on a display, on paper, in speech, or onto other media. The first aspect is called *tree transformation* and the second is called *formatting*. The process of formatting is performed by the formatter. This formatter may simply be a rendering engine inside a browser.

Formatting is enabled by including formatting semantics in the result tree. Formatting semantics are expressed in terms of a catalogue of classes of formatting objects. The nodes of the result tree are formatting objects. The classes of formatting objects denote typographic abstractions such as page, paragraph, table, and so forth. Finer control over the presentation of these abstractions is provided by a set of formatting properties, such as those controlling indents, word- and letter-spacing, and widow, orphan, and hyphenation control. In XSL, the classes of formatting objects and formatting properties provide the vocabulary for expressing presentation intent.

The XSL processing model is intended to be conceptual only. An implementation is not mandated to provide these as separate processes. Furthermore, implementations are free to process the source document in any way that produces the same result as if it were processed using the conceptual XSL processing model. A diagram depicting the detailed conceptual model is shown below:

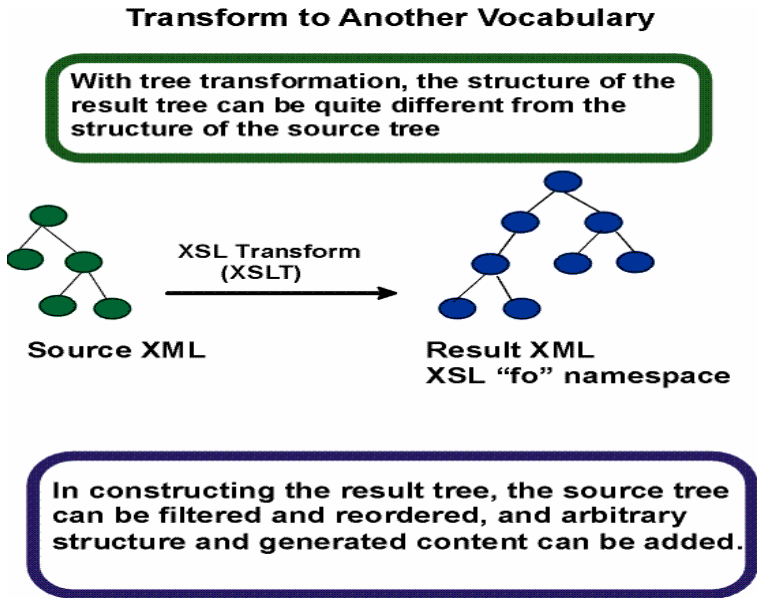


3.6.1 Tree Transformation

Tree transformation constructs the result tree. In XSL, this tree is called the element and attribute tree, with objects primarily in the "formatting object" namespace. In this tree, a formatting object is represented as an XML element, with the properties represented by a set of XML attribute-value pairs. The content of the formatting object is the content of the XML element. Tree transformation is defined in the XSLT Recommendation.

Note also that tree transformation allows the structure of the result tree to be significantly different from the structure of the source tree. For example, one could add a table-of-contents as a filtered selection of an original source document, or one could rearrange source data into a sorted tabular presentation. In constructing the result tree, the tree transformation process also adds the information necessary to format that result tree.

A diagram depicting this conceptual process is shown below:



The XSL style sheet is used in tree transformation. A style sheet contains a set of tree construction rules. The tree construction rules have two parts: a pattern that is matched against elements in the source tree and a template that constructs a portion of the result tree. This allows a style sheet to be applicable to a wide class of documents that have similar source tree structures.

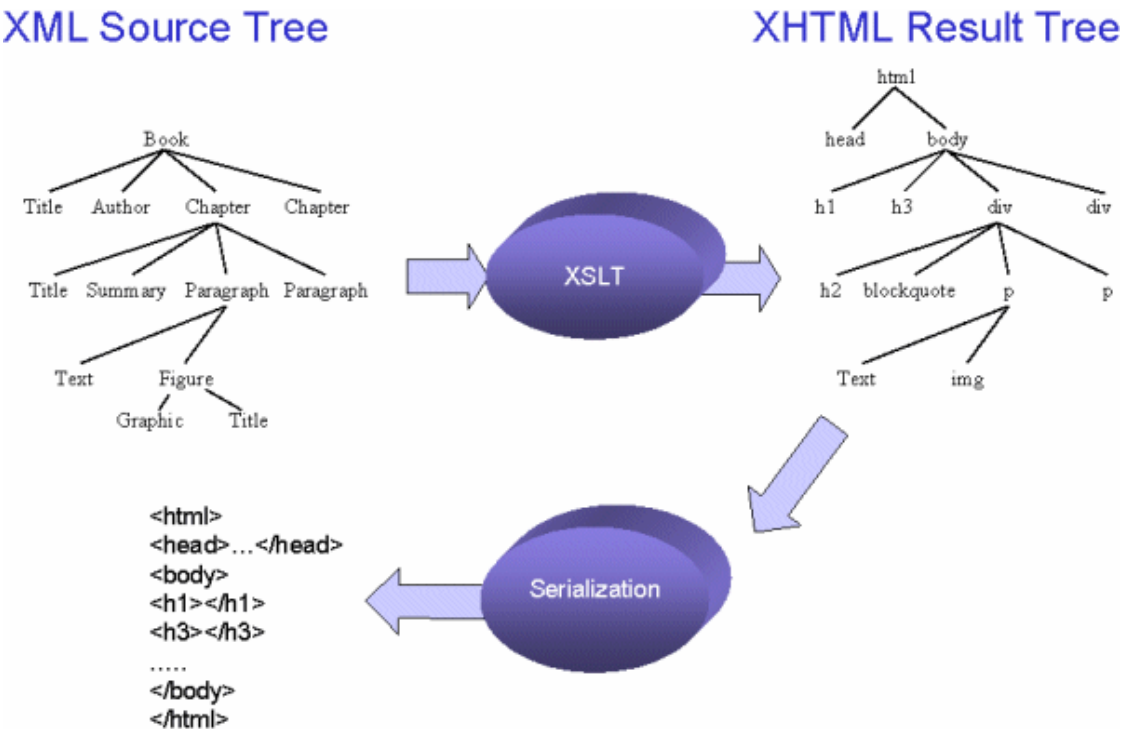


Figure : Tree Transformation

Note that:

- XSLT Style sheets are XML documents; namespaces are used to identify semantically significant elements;
- Most style sheets are stand-alone documents rooted at <xsl:stylesheet> (or <xsl:transform>). It is possible to have "single template" documents/documents; and
- It the mapping from namespace abbreviation to URI that is important, not the literal namespace abbreviation "xsl:" that is used most commonly.

3.6.2 Formatting

Formatting interprets the result tree in its formatting object tree form to produce the presentation intended by the designer of the style sheet from which the XML element and attribute tree in the "fo" namespace was constructed.

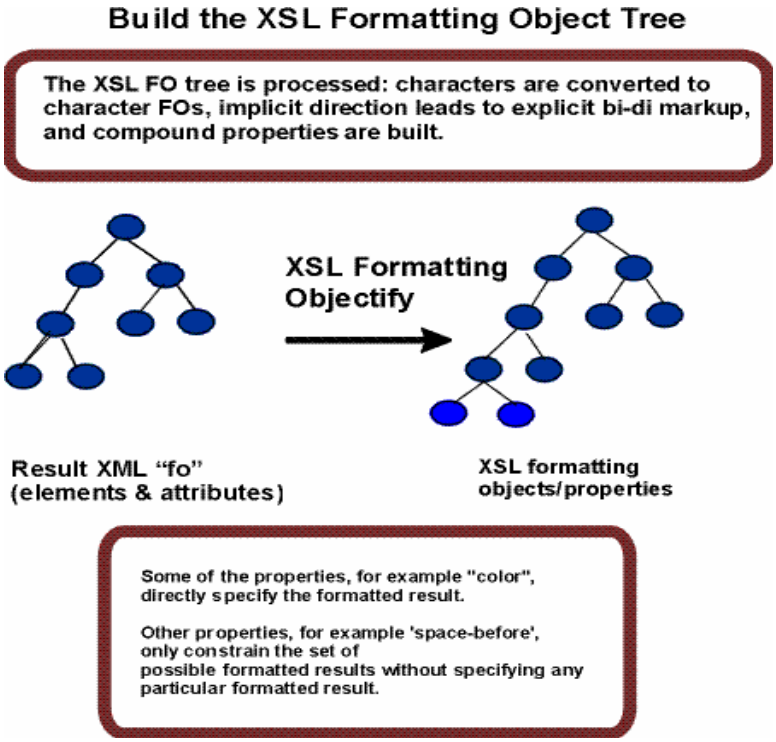
The vocabulary of formatting objects supported by XSL - the set of fo: element types - represents the set of typographic abstractions available to the designer. Semantically, each formatting object represents a specification for a part of the pagination, layout, and styling information that will be applied to the content of that formatting object as a result of formatting the whole result tree. Each formatting object class represents a particular kind of formatting behavior. For example, the block formatting object class represents the breaking of the content of a paragraph into lines. Other parts of the specification may come from other formatting objects; for example, the formatting of a paragraph (block formatting object) depends on both the specification of properties on the block formatting object and the specification of the layout structure into which the block is placed by the formatter.

The properties associated with an instance of a formatting object control the formatting of that object. Some of the properties, for example "color", directly specify the formatted result. Other properties, for example 'space-before', only constrain the set of possible formatted results without specifying any particular formatted result. The formatter may make choices among other possible considerations such as esthetics.

Formatting consists of the generation of a tree of geometric areas, called the area tree. The geometric areas are positioned on a sequence of one or more pages (a browser typically uses a single page). Each geometric area has a position on the page, a specification of what to display in that area and may have a background, padding, and borders. For example, formatting a single character generates an area sufficiently large enough to hold the glyph that is used to present the character visually and the glyph is what is displayed in this area. These areas may be nested. For example, the glyph may be positioned within a line, within a block, within a page.

Rendering takes the area tree, the abstract model of the presentation (in terms of pages and their collections of areas), and causes a presentation to appear on the relevant medium, such as a browser window on a computer display screen or sheets of paper.

The first step in formatting is to "objectify" the element and attribute tree obtained via an XSLT transformation. Objectifying the tree basically consists of turning the elements in the tree into formatting object nodes and the attributes into property specifications. The result of this step is the formatting object tree.

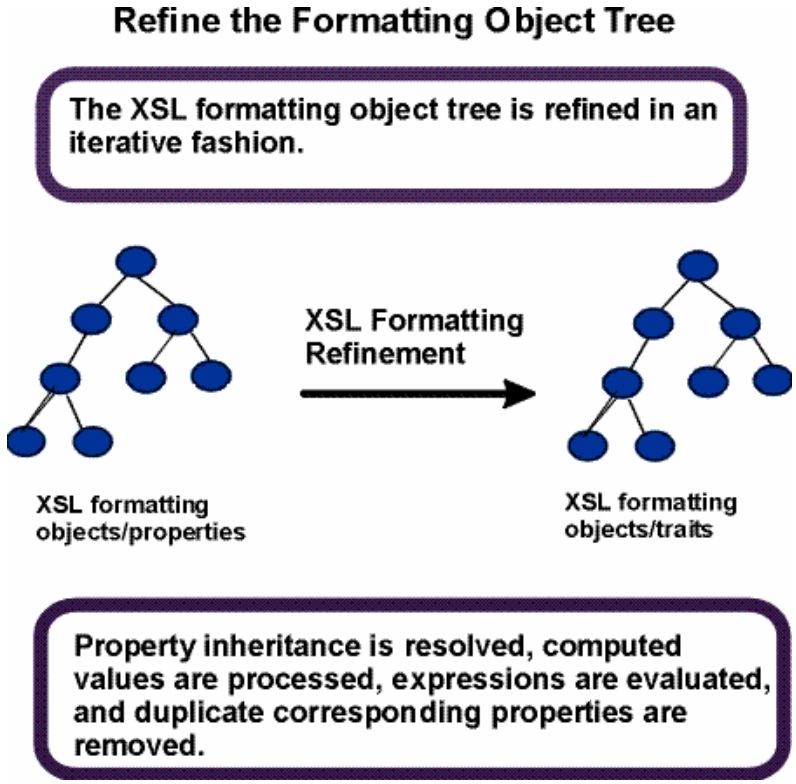


As part of the step of objectifying, the characters that occur in the result tree are replaced by fo:character nodes. The first phase of the Unicode Bi-directional Algorithm is used to convert implicit Bi-directional mark-up to explicit nodes with the appropriate directional properties. Care is taken to insure that the explicit nodes so introduced are properly nested in the formatting object tree.

The second phase in formatting is to refine the formatting object tree to produce the refined formatting object tree. The refinement process handles the mapping from properties to traits. This consists of:

1. shorthand expansion into individual properties;
2. mapping of corresponding properties;
3. determining computed values (may include expression evaluation); and
4. inheritance.

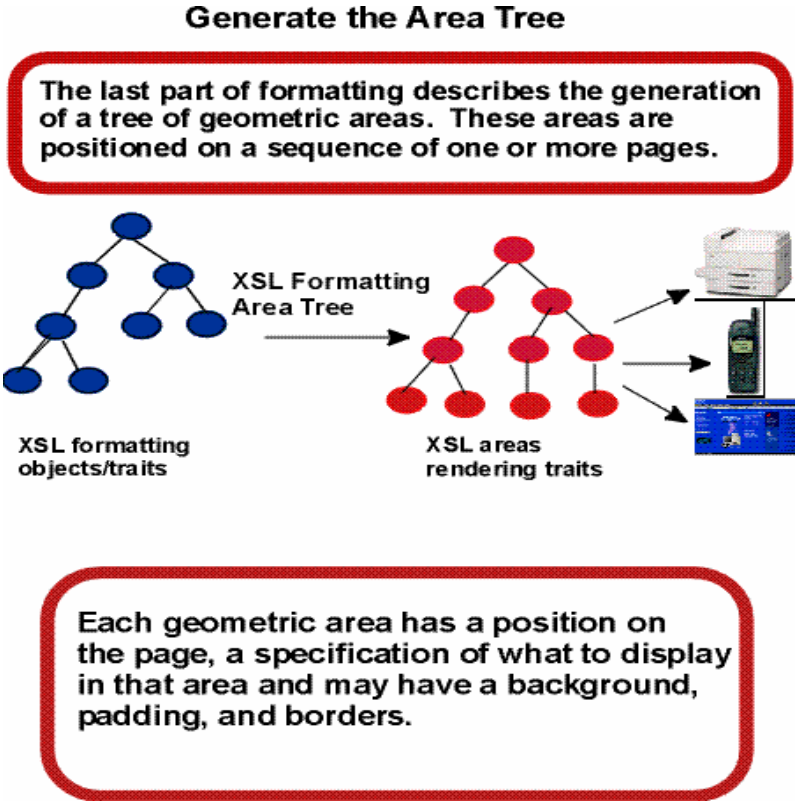
The refinement step is depicted in the diagram below.



3.6.3 Construction Of Area Tree

The third step in formatting is the construction of the area tree. The area tree is generated as described in the semantics of each formatting object. The traits applicable to each formatting object class control how the areas are generated. Although every formatting property may be specified on every formatting object, for each formatting object class, only a subset of the formatting properties are used to determine the traits for objects of that class.

Area generation is depicted in the diagram below:



3.7 Parts Of XSL

XSL consists of three parts, which are fully described in the W3C recommendations. These are (briefly explained here):

- **Xpath**: XML Path language, which is used for referencing parts of an XML document. Although it is not technically XML it is used to address XML document fragments which are portions of an XML document. It is used by XSLT to describe the expressions and location paths that allow creating expressions to manage the selection of nodes for more advanced XSL transformations.

- **XSLT**: XSL Transformations, which is a language that describes the way an XML document can be transformed into another. It transforms the input (source) document's tree into a structure called a result tree consisting of result objects. We can consider of it as a markup vocabulary describing the transformation part of XSL.

- The **XML vocabulary** describing the formatting objects portion of XSL. Formatting objects can be inserted into a result tree created by XSLT (XSL Transformations).

3.8 Formatting Model – The Output of XSL Operation

The transformation process is basically the same no matter what the result tree looks like. Even the IE5-based XSL transformation syntax is very nearly the same thing as the W3C's standards, although the syntactical differences that exist are fatal to a compatible, write-once run-anywhere approach. The formatting model chosen for the result tree, however, can vary significantly. Among the possibilities:

An HTML-based formatting model can be used, meaning we can end up with a result tree consisting of HTML markup (either XML-compliant or not).

We could also use formatting objects based on the W3C's official recommendation for XSL, which are a series of markup tags developed by the W3C that describe a paged media similar to what we might find in print media layout programs but can also be displayed in browsers.

The output of the result tree can be raw text or byte streams or we can output the result into any number of other XML-based vocabularies, such as SVG (Scalable Vector Graphics), an emerging vector graphics Web standard.

Theoretically, the output of the result tree can be PostScript, RTF (Rich Text Format), or some other non-XML language or markup, but there would be a lot of difficulties encountered doing it.

3.9 Structure Of XSL (Using Examples)

A small demo example is following, based on the XSL style sheet presented here:

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

  <xsl:output method="html"/>

  <xsl:template match="doc">
    <html>
      <head><title><xsl:value-of select="title"/></head>
      <body><xsl:apply-templates/></body>
    </html>
  </xsl:template>

  <xsl:template match="title">
    <h1><xsl:apply-templates/></h1>
  </xsl:template>
```

```
  <xsl:template match="para">
    <p><xsl:apply-templates/></p>
  </xsl:template>
</xsl:stylesheet>
```

3.9.1 Templates

- **Understanding a template**

Most templates have the following form:

```
<xsl:template match="para">
  <p><xsl:apply-templates/></p>
</xsl:template>
```

Explanation of the template:

- The whole <xsl:template> element is a template
- The match pattern determines where this template applies
- Literal result elements come from non-XSL namespace(s)
- XSLT elements come from the XSL namespace

- **The Application of templates is Transformation**

Templates transform portions of the source tree into portions of the result tree. The ordered accumulation of all the transformed portions forms the complete result tree. Individual templates are free to process elements from anywhere in the source tree.

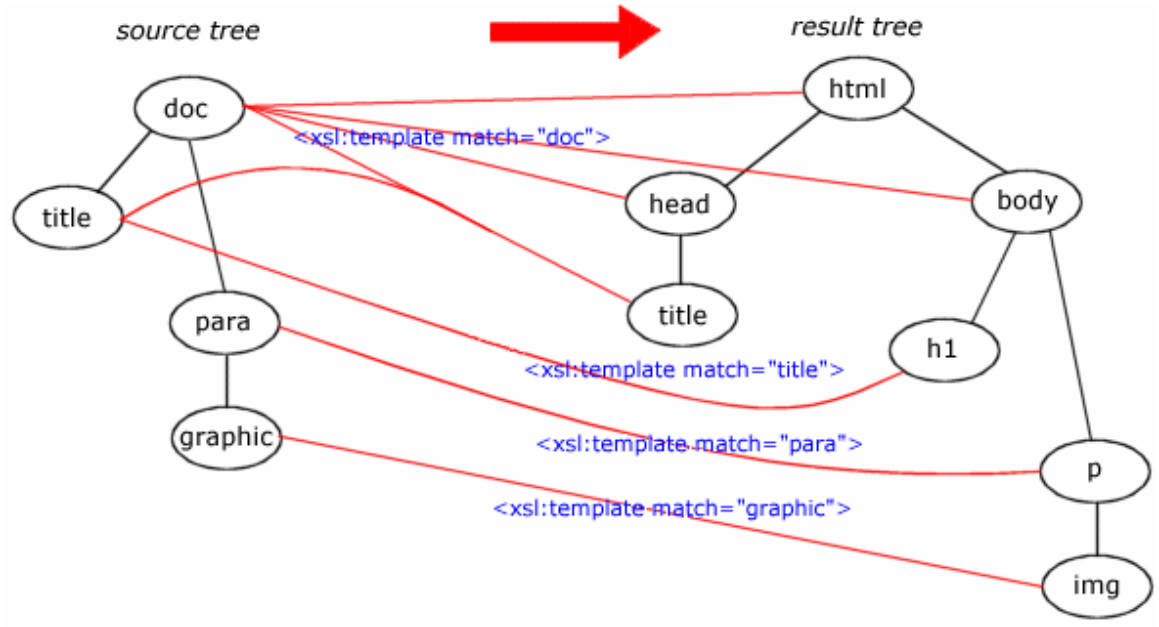


Figure : Transformation from Source Tree to Result Tree

- **Match Patterns (Locating Elements)**

One critical capability of a style sheet language is to locate source elements to be styled. CSS, for example, does this with "selectors." FOSLs (which is another style sheet language) perform it using "e-i-c's" elements in context. XSLT accomplishes it with "match patterns" defined by XPath.

XPath has an extensible string-based syntax inspired, in part, by the common "path/file" file system syntax:

EXAMPLES:

- para : Matches all <para> children in the current context
- para/emphasis : Matches all <emphasis> elements that have a parent of <para>
- ancestor-or-self::*/@sepchar : Matches the sepchar attribute on the current element or any ancestor of the current element
- numberedlist/listitem[position() mod 2 = 0] : Matches odd list items in a numbered list.

- **Applying Style Recursively**

One model for applying style is to allow the process to run recursively, driven primarily by the document. A series of templates is created, such that there is a template to match each context, then these templates are recursively applied starting at the root of the document.

<xsl:template match="...">

e.g. <xsl:template match="section/title">

<h2><xsl:apply-templates/></h2>

</xsl:template>

<xsl:apply-templates>

e.g. <xsl:apply-templates select="th|td"/>

NOTE: There are two obstacles to overcome when using the recursive model, how to arbitrate between multiple patterns that match and how to process the same nodes in different contexts. These are solved by conflict resolution and modes, respectively.

- **Applying Style Procedurally**

The other model for applying style is to select each action procedurally. A series of templates is created, such that each template explicitly selects and processes the necessary elements.

- **<xsl:for-each>**

e.g. <xsl:for-each select="row">

<tr>

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

<td><xsl:value-of select="."/></td>

</tr>

</xsl:for-each>

- **<xsl:template name="...">**

e.g. <xsl:template name="admonition">

<xsl:param name="type">warning</xsl:param>

```
  ...
</xsl:template>
```

- **<xsl:call-template>**

```
e.g. <xsl:call-template name="admonition">
  <xsl:with-param name="type">caution</xsl:with-param>
</xsl:call-template>
```

3.9.2 Conditional Processing

- **<xsl:if>**

```
e.g. Simple conditional (no "else")
<xsl:if test="{$somecondition}">
  <xsl:text>this text only gets used if $somecondition is true()</xsl:text>
</xsl:if>
```

- **<xsl:choose>**

```
e.g. Select among alternatives with <xsl:when> and <xsl:otherwise>
<xsl:choose>
  <xsl:when test="$count > 2"><xsl:text>, and </xsl:text></xsl:when>
  <xsl:when test="$count > 1"><xsl:text> and </xsl:text></xsl:when>
  <xsl:otherwise><xsl:text> </xsl:text></xsl:otherwise>
</xsl:choose>
```

3.9.3 Variables

- Variables can be used to save computed values;
- Variables are created with **<xsl:variable>**;
- Variables are "single assignment" (no side effects); and
- Variables are lexically scoped.

Once created, variables can be used to generate content:

e.g. ...

And control conditional processing:

e.g. <xsl:if test="$count = 3">...</xsl:if>

3.9.4 Creating the Result Tree

- **Literal Result Elements**

Any element in a template rule that is not in the XSL (or other extension) namespace is copied literally to the result tree: ...

- **XSL Elements**

Elements in the XSL namespace:

  - <xsl:text>
  - <xsl:value-of>
  - <xsl:element>
  - <xsl:attribute>

- **Numbering and Sorting**

It is also possible to:

  - Count source tree elements (chapters, list-items, stock quotes, etc.);
  - Convert between number formats (1, B, iii, ...); and
  - Sort elements for presentation

3.9.5 Overall XSL formatting capabilities

XSL FO formatting capabilities in XSL 1.0 are approximately the union of:

- HTML + CSS capabilities; and
- most high quality print output capabilities including internationalization features.

Not included are complex page layouts (e.g., magazine and newspaper layout), complex layout-driven formatting (e.g., copy fitting and complex floats), and loose-leaf pagination (change page production)

3.9.6 Formatting objects and properties

- XSL = XSLT + vocabulary of FOs and properties;
- XSL defines a powerful set of formatting objects; and
- XSL uses (and extends) a set of Common Formatting Properties developed jointly with the CSS&FP (Cascading Style Sheet and Formatting Property) Working Group.

When a result tree uses this standardized set of formatting objects and properties, then an XSL-compliant formatter can process that result tree to produce the specified output.

3.9.7 Formatting Object Basics

- Inline versus block objects;
- Common formatting properties--harmonized with CSS.

Common Formatting Objects

- page-sequence--a major part (such as front or body) in which the basic page layout may differ from other parts;
- flow--a chapter- or section-like division within a page-sequence;
- block--a paragraph (or title or block quote, etc.);
- inline--e.g., a font change within a paragraph;
- wrapper--a "transparent" object usable as either a block or inline object that has no effect other than to provide a place to hang inheritable properties;
- list FOs--list-block, list-item, list-item-label, list-item-body;
- graphic--references an external graphic object; and
- table FOs--mostly analogous to the standard (CALS, OASIS, HTML) table models.

3.9.8 Basic properties

- font properties;
- margin and spacing properties;
- border and padding properties;
- keeps/breaks;
- horizontal alignment/justification;

- indentation; and
- more formatting object specific properties.

3.10 Benefits of XSL

Unlike the case of HTML, element names in XML have no intrinsic presentation semantics. Absent a style sheet, a processor could not possibly know how to render the content of an XML document other than as an undifferentiated string of characters. XSL provides a comprehensive model and a vocabulary for writing such style sheets using XML syntax.

XSL builds on the prior work on Cascading Style Sheets(CSS) and the Document Style Semantics and Specification Language (DSSSL). While many of XSL's formatting objects and properties correspond to the common set of properties, this would not be sufficient by itself to accomplish all the goals of XSL. In particular, XSL introduces a model for pagination and layout that extends what is currently available and that can in turn be extended, in a straightforward way, to page structures beyond the simple page models existing today.

1. Paging and Scrolling

Doing both scrollable document windows and pagination introduces new complexities to the styling (and pagination) of XML content. Because pagination introduces arbitrary boundaries (pages or regions on pages) on the content, concepts such as the control of spacing at page, region, and block boundaries become extremely important. There are also concepts related to adjusting the spaces between lines (to adjust the page vertically) and between words and letters (to justify the lines of text). These do not always arise with simple scrollable document windows, such as those found in today's browsers. However, there is a correspondence between a page with multiple regions, such as a body, header, footer, and left and right side-bars, and a Web presentation using "frames". The distribution of content into the regions is basically the same in both cases, and XSL handles both cases in an analogous fashion.

XSL was developed to give designers control over the features needed when documents are paginated as well as to provide an equivalent "frame" based structure for browsing on the Web. To achieve this control, XSL has extended the set of formatting objects and formatting properties.

In addition, the selection of XML source components that can be styled (elements, attributes, text nodes, comments, and processing instructions) is based on XSLT and XPath, thus providing the user with an extremely powerful selection mechanism.

The design of the formatting objects and properties extensions was first inspired by DSSSL. The actual extensions, however, do not always look like the DSSSL constructs on which they were based. To either conform more closely with the CSS2 specification or to handle cases more simply than in DSSSL, some extensions have diverged from DSSSL.

There are several ways in which extensions were made. In some cases, it sufficed to add new values, as in the case of those added to reflect a variety of writing-modes, such as top-to-bottom and bottom-to-top, rather than just left-to-right and right-to-left.

In other cases, common properties that are expressed in CSS2 as one property with multiple simultaneous values, are split into several new properties to provide independent control over independent aspects of the property. For example, the "white-space" property was split into four properties: a "space-treatment" property that controls how white-space is processed, a "line-feed" property that controls how line-feeds are processed, a "white-space-collapse" property that controls how multiple consecutive spaces are collapsed, and a "wrap-

option" property that controls whether lines are automatically wrapped when they encounter a boundary, such as the edge of a column. The effect of splitting a property into two or more (sub-)properties is to make the equivalent existing CSS2 property a "shorthand" for the set of sub-properties it subsumes.

In still other cases, it was necessary to create new properties. For example, there are a number of new properties that control how hyphenation is done. These include identifying the script and country the text is from as well as such properties as "hyphenation-character" (which varies from script to script).

Some of the formatting objects and many of the properties in XSL come from the CSS2 specification, ensuring compatibility between the two.

There are four classes of XSL properties that can be identified as:

- CSS properties by copy (unchanged from their CSS2 semantics);
- CSS properties with extended values;
- CSS properties broken apart and/or extended; and
- XSL only properties.

2. Selectors and Tree Construction

As mentioned above, XSL uses XSLT and XPath for tree construction and pattern selection, thus providing a high degree of control over how portions of the source content are presented, and what properties are associated with those content portions, even where mixed namespaces are involved.

For example, the patterns of XPath allow the selection of a portion of a string or the Nth text node in a paragraph. This allows users to have a rule that makes all third paragraphs in procedural steps appear in bold, for instance. In addition, properties can be associated with a content portion based on the numeric value of that content portion or attributes on the containing element. This allows one to have a style rule that makes negative values appear in "red" and positive values appear in "black". Also, text can be generated depending on a particular context in the source tree, or portions of the source tree may be presented multiple times with different styles.

3. An Extended Page Layout Model

There is a set of formatting objects in XSL to describe both the layout structure of a page or "frame" (how big is the body; are there multiple columns; are there headers, footers, or side-bars; how big are these) and the rules by which the XML source content is placed into these "containers".

The layout structure is defined in terms of one or more instances of a "simple-page-master" formatting object. This formatting object allows one to define independently filled regions for the body (with multiple columns), a header, a footer, and side-bars on a page. These simple-page-masters can be used in page sequences that specify in which order the various simple-page-masters shall be used. The page sequence also specifies how styled content is to fill those pages. This model allows one to specify a sequence of simple-page-masters for a book chapter where the page instances are automatically generated by the formatter or an explicit sequence of pages such as used in a magazine layout. Styled content is assigned to the various regions on a page by associating the name of the region with names attached to styled content in the result tree.

In addition to these layout formatting objects and properties, there are properties designed to provide the level of control over formatting that is typical of paginated documents. This includes control over hyphenation, and expanding the control over text that is kept with other text in the same line, column, or on the same page.

4. A Comprehensive Area Model

The extension of the properties and formatting objects, particularly in the area on control over the spacing of blocks, lines, and page regions and within lines, necessitated an extension of the CSS2 box formatting model. The CSS2 box model is a subset of this model.The area model provides a vocabulary for describing the relationships and space-adjustment between letters, words, lines, and blocks.

5. Internationalization and Writing-Modes

There are many scripts, in particular in the Far East, that are typically set with words proceeding from top-to-bottom and lines proceeding either from right-to-left (most common) or from left-to-right. Other directions are also used. Properties expressed in terms of a fixed, absolute frame of reference (using top, bottom, left, and right) and which apply only to a notion of words proceeding from left to right or right to left do not generalize well to the languages based on these scripts.

For this reason XSL (and before it DSSSL) uses a relative frame of reference for the formatting object and property descriptions. Just as the CSS2 frame of reference has four directions (top, bottom, left and right), so does the XSL relative frame of reference have four directions (before, after, start, and end), but these are relative to the "writing-mode". The "writing-mode" property is a way of controlling the directions needed by a formatter to correctly place glyphs, words, lines, blocks, etc. on the page or screen. The "writing-mode" expresses the basic directions noted above. There are writing-modes for "left-to-right - top-to-bottom" (denoted as "lr-tb"), "right-to-left - top-to-bottom" (denoted as "rl-tb"), "top-to-bottom - right-to-left" (denoted as "tb-rl") and more. Typically, the writing-mode value specifies two directions, the first is the inline-progression-direction which determines the direction in which words will be placed and the second is the block-progression-direction which determines the direction in which blocks (and lines) are placed one after another.

Besides the directions that are explicit in the name of the value of the "writing-mode" property, the writing-mode determines other directions needed by the formatter, such as the shift-direction (used for sub- and super-scripts), etc.

6. Linking

Because XML, unlike HTML, has no built-in semantics, there is no built-in notion of a hypertext link. Therefore, XSL has a formatting object that expresses the dual semantics of formatting the content of the link reference and the semantics of following the link.

XSL provides a few mechanisms for changing the presentation of a link target that is being visited. One of these mechanisms permits indicating the link target as such; another allows for control over the placement of the link target in the viewing area; still another permits some degree of control over the way the link target is displayed in relationship to the originating link anchor.

XSL also provides a general mechanism for changing the way elements are formatted depending on their active state. This is particularly useful in relation to links, to indicate whether a given link reference has already been visited, or to apply a given style depending on whether the mouse, for instance, is hovering over the link reference or not.

3.11 XSL-Enabled Tools

3.11.1 XSLT Processors

- 4XSLT is an XML transformation processor written in Python that implements the XSLT transform language. (http://fourthought.com/4Suite/4XSLT)

- The InDelv browser implements XSL style sheets, including the FO part for direct display. It also implements XLink. (http://www.indelv.com)
- XSL is integrated into the Microsoft XML processor which is part of Internet Explorer 5. It transforms XML into HTML, which is then displayed using CSS; it does not implement FOs.
- iXSLT from Infoteria is a XSLT processor written in C++. (www.infoteria.com/en/contents/download/index.html)
- LotusXSL is a complete implementation of the W3C Recommendations for XSL Transformations (XSLT) and the XML Path Language (XPath).(http://www.alphaworks.ibm.com/tech/LotusXSL)
- Transformiix is an XSLT processor in C++ available from Mozilla.org. (http://lxr.mazilla.org/mozilla/source/extensions/transformiix)
- Resin is a servlet/JSP engine with integrated XPath and XSLT support. (http://www.caucho.com/products/resin/index.html)
- Sablotron is an attempt to develop a fast, compact and portable XSLT processor written in C++. (http://www.gigerall.com/charlie-bin/get/webGA/act/sablotron.act)
- Saxon is a collection of tools for processing XML documents. It includes a complete implementation of the XSLT 1.0 and XPath 1.0 Recommendations, as well as a Java library. (http://users.iclway.co.uk/mhkay/saxon/)
- Xalan is a full implementation of the W3C recommendations XSLT and XPath. It's provided by the Apache XML Project. (http://xml.apache.org/xalan/overview.html)
- The XML Parser for Java v.2 from Oracle incorporates support for XSL Transformations (XSLT). (http://technet.oracle.com/tech/xml)
- XMLwriter is an XML editor that supports XSL, so you can transform the content and style of your XML documents. (http://xmlwriter.net/index.html)
- XT from James Clark is a free Java-based implementation of XSLT.(http://jclark.com/xml/xt.html)

3.11.2 XSL-FO processors

- FOP is a XSL-FO to PDF converter developed by James Tauber at the Apache Software FoundationXE (http://xml.apache.org/fop)
- Passive TeX is a library of TeX macros which provides a rapid development environment for experimenting with XSL-FO.(http://users.ox.ac.uk/~rahtz/passivetex/)
- REXP is an early implementation of a Formatting Objects engine based on FOP. It generates PDF files. It's an open source.(http://www.esng.dibe.unige.it/REXP)
- XEP (formerly known as FOP2PDF) from RenderX is a program for converting XSL-FO documents to PDF.(http://www.renderx.com/FO2PDF.html)

3.11.3 XSL-Enabled Authoring Tools

- eXcelon Stylus (http://www.exceloncorp.com/products/excelon-stylus.html) combines tools to create XSL style sheets in a visual editing environment.

  The eXtensible Style sheet Language (XSL) is the key ingredient to making XML work for eBusiness.

  A member of the XML family of standards, XSL was developed by the World Wide Web Consortium (W3C) for dynamically transforming XML content. Using XSL, you can transform XML into other XML dialects or into HTML with a simple, rules-based language.

  eXcelon Stylus is the first integrated environment for creating, managing and maintaining XSL style sheets. With Stylus, you can quickly and easily create style sheets that let you fully leverage all of your corporate information. Stylus speeds initial development with its built-in knowledge of XSL commands, and eases maintenance through its intuitive one-click debugging techniques.

  eXcelon Stylus makes XML work for eBusiness.

- The IBM XSL Editor application allows a user to import, create, and save XSL style sheets and XML source documents.(http://www.alphaworks.ibm.com/tech/xsleditor)

There are two screenshots, coming from the execution of the software, presented above, just to get an idea of the interface and simplicity these tool offers.

Figure : The Stylus three-pane editing environment with Sense:X automatic tag completion

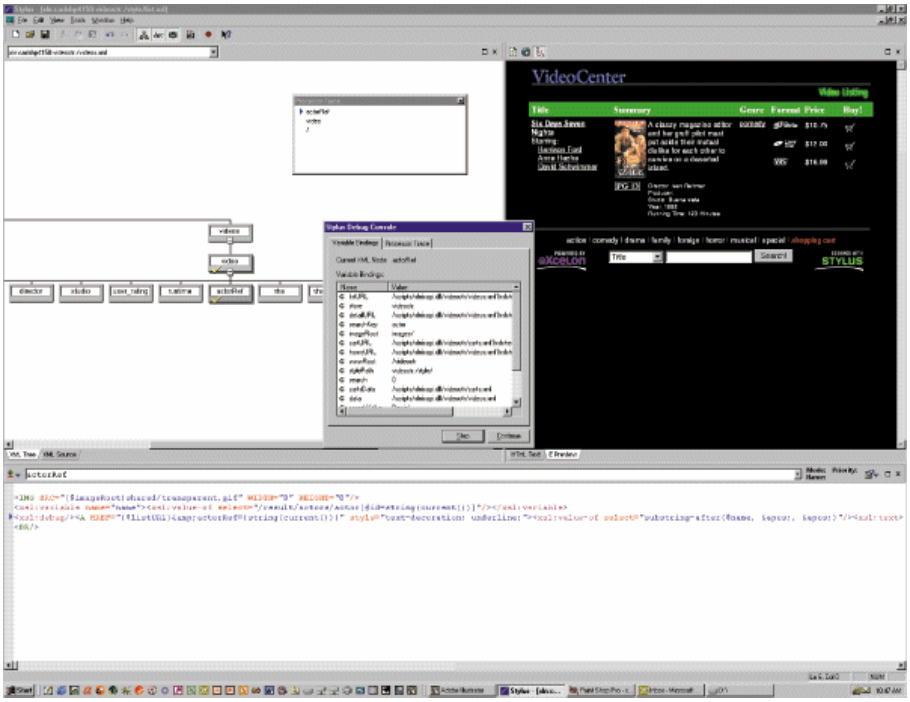
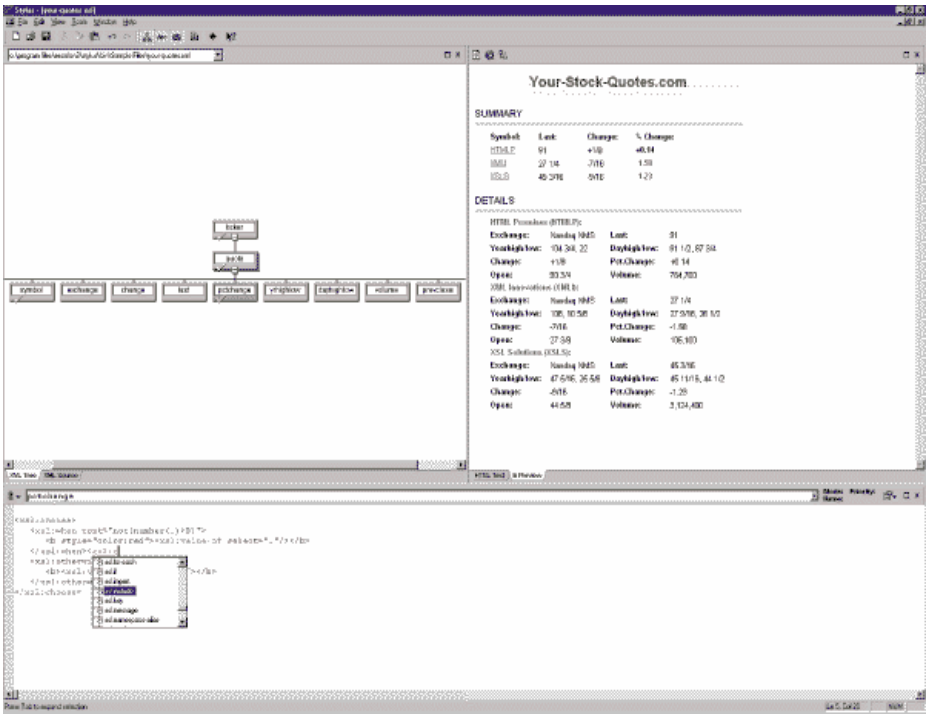


Figure: The Stylus Processor Trace and Debug Console make style sheet editing painless



3.12 The Future Of XSL

XSL is based on DSSSL and is compatible with the fundamental design principles and processing model of the DSSSL standard. However, the development of XSL has identified usability issues with the current DSSSL standard which has led XSL to diverge in various ways.

Therefore, in parallel with the development of the XSL specification, a proposal for an amendment to DSSSL will be developed so that with the amendment it would be a superset of XSL. The amended standard would be compatible with current DSSSL but would have three kinds of additions:

- an alternative syntax compatible with the XSL syntax;
- extensions to the flow object tree construction language to support the features of XSL not in DSSSL (these features would be available both with the current syntax and the alternative syntax); and
- new flow object classes and characteristics to support formatting functionality necessary for the Web.

4. XQL Language

At present time, as more and more information is either stored in XML, exchanged in XML, or presented as XML through various interfaces, the ability of intelligent query of XML data sources becomes increasingly important. Besides, it is widely believed that XML can solve the search problem. The "tool" which promises the success of querying of XML documents is XQL, which stands for XML Query Language.

XQL's first draft was written in February 1998 after several months of development and is still under further construction and study. It is implemented by W3C organization.

4.1 Why Use A Query Language?

Traditionally, structured queries have been used primarily for relational or object oriented databases, and documents were queried with relatively unstructured full-text queries. Although quite sophisticated query engines for structured documents have existed for some time, they have not been a mainstream application.

Queries within a single document are useful in XML browsers or editors to allow the user to query large documents and find relevant information without scrolling through the entire document. The XML document upon a query is applied, is called *search context*. As a matter of fact**,** the search context is the set of XML document nodes for which the query is evaluated. A query engine executes an XQL query for a given search context, which may be the set of nodes at the root of a document, or any other set of nodes made available to the query environment.

4.2 Purposes Of Creating XQL

While designing the XQL language four problem domains were chosen to determine the requirements:

- Queries within a single document;
- Queries in a collection of documents;
- Addressing within or across documents; and
- XSL Patterns.

These are actually quite distinct problem domains, and not all of these are queries in the traditional sense of the term – XPointers identify known locations in specific documents, XSL patterns are often evaluated for specific elements that are being processed in order to determine whether a template applies, and queries determine whether data that meets specific conditions are present, returning the relevant data if found. Because of the notable differences in the tasks, the SGML and XML communities have traditionally treated these as completely separate domains.

There is, however, a great deal of commonality among these tasks, especially with respect to their need to specify XML structures. The processing that is done may be quite different in the different domains; still, the needs of each are quite similar on the query-language level. In some ways this is analogous to the differing uses of SQL in traditional relational databases. SQL may be used in a variety of ways that involve different

forms of processing, e.g. to establish views, execute queries, or set cursors, but the query language used to perform these various tasks is the same.

Various problem domains also show that they vary widely in terms of their input, output, processing model, and the desired form in which results are returned. However, they all have one thing in common: they need to be able to specify one or more nodes based on assertions about their names, content, values, and relationships to other nodes (whose names, content, and values may also be specified). XQL is a means of specifying these assertions. Since XQL says nothing about the manner in which input is provided, the format of the output, or the processing model used to apply these assertions, it is completely independent of the factors which distinguish these problem domains.

Finally, the input to an XQL query is an entire document, which may be of any size. A sophisticated implementation may have indexes defined for the document, and may have ways to avoid loading the entire document into memory; a relatively naive implementation may simply search the entire document. The result of such a query may differ among implementations; e.g., the result may be an iterator that traverses precisely those nodes returned by the query, in document order, or it may be a set of addresses that may be used to jump to the appropriate position in the original document, or it may be a navigation object that allows the nodes returned by the query to be traversed as a virtual tree, i.e. a view containing a subset of the original document. The subject of where can the results be stored, is discussed later in this report.

Conclusively, although the manner in which results are retrieved and returned may differ considerably, the role of the query language is the same in all of these cases: it must describe the set of nodes that should be returned.

4.3 SQL vs. XQL

It would be useful, in order to understand better the usability and difference of the XML query language, to compare it with an other, very popular, query language, the structured query language (SQL):

SQL	XQL
DATABASE The database is a set of tables.	DATABASE The database is a set of one or more XML documents.
USED MODEL Queries are done in SQL, a query language that uses tables as a basic model.	USED MODEL Queries are done in XQL, a query language that uses the structure of XML as a basic model.
EXAMINED ENTITIES The FROM clause determines the tables which are examined by the query.	EXAMINED ENTITIES A query is given a set of input nodes from one or more documents, and examines those nodes and their descendants.
RESULT The result of a query is a table containing a set of rows.	RESULT The result of a query is a set of XML document nodes, which can be wrapped in a root node to create a well-formed XML document.

It is obvious that XQL and SQL are similar to the way they "think". They just want to extract the needed data from a "data source" using several criteria. But, XQL is totally based on XML. XML documents are structured documents, which can be treated like data sources (and data sources can be treated as documents). XQL takes full advantage of it. The input consists of a XML document and the output can also be an XML document.

XML takes the table's place in SQL, because now, data can be stored in documents.

4.4 Where Can XQL Queries Be Used?

Queries may be used in various environments and represent a wide range of activities and needs that are representative of the problem space to be addressed. Thus, they may be used in scripting languages to provide powerful non-procedural access to document data and structures. In addition, they may be used by document authors to define various views of a document, e.g. for users with varying background, or users with differing access rights. Some other, better defined usage scenarios include the following.

1) Human-readable documents

Perform queries on structured documents and collections of documents, such as technical manuals, to retrieve individual documents, to generate tables of contents, to search for information in structures found within a document, or to generate new documents as the result of a query.

2) Data-oriented documents

Perform queries on the XML representation of database data, object data, or other traditional data sources to extract data from these sources, to transform data into new XML representations, or to integrate data from multiple heterogeneous data sources. The XML representation of data sources may be either physical or virtual; that is, data may be physically encoded in XML, or an XML representation of the data may be produced.

3) Mixed-model documents

Perform both document-oriented and data-oriented queries on documents with embedded data, such as catalogues, patient health records, employment records, or business analysis documents.

4) Administrative data

Perform queries on configuration files, user profiles, or administrative logs represented in XML.

5) Filtering streams

Filter streams of XML data, such as logs of email messages, network packets, stock market data, newswire feeds, EDI, or weather data, either as a traditional UNIX-style filter that extracts or transforms its input data, or to specify filters and profiles for routing messages represented in XML.

6) Document Object Model (DOM)

Perform queries on DOM structures to return sets of nodes that meet the specified criteria.

7) Native XML repositories and web servers

Perform queries on collections of documents managed by native XML repositories or web servers.

8) Catalogue search

Perform queries to search catalogues that describe document servers, document types, or documents. Such catalogues may be combined to support search among multiple servers. A document-retrieval system could use queries to allow the user to select server catalogues, represented in XML, by the information provided by the servers, by access cost, or by authorisation. Once a server is selected, a retrieval system could query the kinds of documents found on the server and allow the user to query those documents.

9) Multiple syntactic environments

Queries may be used in many environments. For example, a query might be embedded in a URL, an XML page, or a JSP or ASP page; represented by a string in a program written in a general-purpose programming language; provided as an argument on the command-line or standard input; or supported by a protocol, such as DASL (DAV Searching and Locating) or Z39.50.

4.5 General Requirements Of XQL

During the implementation of the XQL, the following general requirements were decided to be taken under consideration [XML Query Requirements -W3C Working Draft 31 January 2000]:

- **Query Language Syntax**

The XML Query Language may have more than one syntax binding. One query language syntax must be convenient for humans to read and write. One query language syntax must be expressed in XML in a way that reflects the underlying structure of the query.

- **Declarativity**

The XML Query Language must be declarative. Notably, it must not enforce a particular evaluation strategy.

- **Protocol Independence**

The XML Query Language must be defined independently of any protocols with which it is used.

The XML Query Language must define standard error conditions that can occur during the execution of a query, such as processing errors within expressions, unavailability of external functions to the query processor, or processing errors generated by external functions.

- **Updates**

The XML Query Language must not preclude the ability to add update capabilities in future versions.

- **Defined for Finite Instances**

The XML Query Language must be defined for finite instances of the data model. It may be defined for infinite instances.

4.5.1 XML Query Data Model

- **Reliance on XML Information Set**

The XML Query Data Model relies on information provided by XML Processors and Schema Processors, and it must ensure that it does not require information that is not made available by such processors. For

XML constructs found in XML 1.0 or the Namespaces Recommendation, the XML Query Data Model must show how the equivalent XML Query Data Model constructs are defined in terms of items in the XML Information Set. The XML Query Data Model should represent all information items, or provide justification for any information items omitted. For information found in the XML Schema, such as data types, the Data Model must coordinate with the XML Schema Working Group to ensure that schema processors may be relied on to provide the information needed to construct the Data Model.

- **Datatypes**

The XML Query Data Model must represent both XML 1.0 character data and the simple and complex types of the XML Schema specification.

- **Collections**

The XML Query Data Model must represent collections of documents and collections of simple and complex values. (Note that collections are not part of the current XML Infoset.)

- **References**

The XML Query Data Model must include support for references, including both references within an XML document and references from one XML document to another.

- **Schema Availability**

Queries must be possible whether or not a schema is available (in this document, the term "schema" may refer to either an XML Schema or a DTD). If a schema is available, the data model must represent any items which they define for their instances, such as default attributes, entity expansions, or data types. These items will not be present if a schema is not present.

- **Namespace Awareness**

The XML Query Language and XML Query Language Data Model must be namespace aware.

4.5.2 XML Query Functionality

- **Supported Operations**

  The XML Query Language must support operations on all data types represented by the XML Query Data Model.

- **Text and Element Boundaries**

  Operations on text must be applicable to text that spans element boundaries.

- **Universal and Existential Quantifiers**

  Operations on collections must include support for universal and existential quantifiers.

- **Hierarchy and Sequence**

  Queries must support operations on hierarchy and sequence of document structures.

- **Combination**

  The XML Query Language must be able to combine related information from different parts of a given document or from multiple documents.

- **Aggregation**

  The XML Query Language must be able to compute summary information from a group of related document elements (this operation is sometimes called "aggregation.")

- **Composition of Operations**

  The XML Query Language must support expressions in which operations can be composed, including the use of queries as operands.

- **NULL Values**

  The XML Query Language must include support for NULL values. Therefore, all operators must take NULL values into account, including logical operators.

- **Structural Preservation**

  Queries must be able to preserve the relative hierarchy and sequence of input document structures in query results.

- **Structural Transformation**

  Queries must be able to transform XML structures and must be able to create new structures.

- **References**

  Queries must be able to traverse intra- and inter-document references.

- **Identity Preservation**

  Queries must be able to preserve the identity of items in the XML Query Data Model.

- **Operations on Literal Data**

  Queries should be able to operate on XML Query Data Model instances specified with the query. We refer to such data as "literal data" in this document.

- **Operations on Names**

  Queries should be able to operate on names, such as element names, attribute names, and processing instruction targets, and to operate on combinations of names and data.

- **Operations on Schemas**

  Queries should provide access to the XML schema or DTD for a document, if there is one. If the schema is represented as a DTD, a mapping to an appropriate XML Schema representation may be required.

- **Extensibility**

  The XML Query Language should support the use of externally defined functions on all data types of the XML Query Data Model. The interface to such functions should be defined by the Query Language, and should distinguish these functions from functions defined in the Query Language. The implementation of externally defined functions is not part of the Query Language.

- **Environment Information**

  The XML Query Language must provide access to information derived from the environment in which the query is executed, such as the current date, time, or user.

- **Closure**

  Queries must be closed with respect to the XML Query Data Model. Both the input to a query and the output of a query must be defined purely in terms of the XML Query Data Model. Non-XML sources such as traditional databases or objects may be queried if they are given an XML Query Data Model

representation. Similarly, query results are defined purely in terms of the XML Query Data Model. In software systems these results may be instantiated in any convenient representation such as DOM nodes, hyperlinks, XML text, or various data formats.

NOTE: The keywords “must”, ‘’must not’’, “may”, “should” and “should not”, which are used for expressing the above requirements, have the following meaning [defined in RFC2119]:

1. **MUST:** This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

2. **MUST NOT:** This phrase, or the phrase "SHALL NOT", mean that the definition is an absolute prohibition of the specification.

3. **SHOULD:** This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

4. **SHOULD NOT:** This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.

5. **MAY:** This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option MUST be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

4.6 Criteria Used To Extract Data

The criteria that are used to describe the nodes (which represent data in a XML document) we are looking for can be:

- The name of any element or attribute, or the target of a processing instruction;
- The type of a node (element, attribute, processing instruction, comment, etc.);
- The content or the value of any node; and
- Relationships among nodes identified by the above criteria, including hierarchy, sequence, and position.

4.7 Storing The Results Of An XQL Query

XQL specifies nothing about the physical representation of the result. The result may be represented in a variety of ways, e.g. as ASCII text, as Document Object Model nodes, as a set of hypertext links, by setting an internal cursor, or by executing a procedure to process the selected nodes.

The ***result set*** is the set of nodes selected by the query. Since many environments will want to process the result set further, e.g. by transforming or formatting it with XSL, XQL represents results as a well-formed XML document called the result of the query. Since XML documents may have only one root node, the result of a query wraps the result set in an <xql:result> element. This is the only difference between the result and the result set. There are some query expressions that return text that can not be converted to a well-formed document simply by placing the result set in an element wrapper; e.g., when attributes are returned without the element that contains them.

Consequently, the main reasons for storing the query results in a well-formed XML document include:

- An XML document is easily parsed with a standard XML parser, so it can be transmitted as a single ASCII stream and parsed by the receiving application;
- An XML document can be displayed in a standard XML browser;
- An XML document can be stored in an XML repository; and
- An XML document can be passed on to an XSL processor to perform transformations or do formatting.

4.8 Basic Syntax of XQL

XQL expressions are easily parsed, easy to type and can be used in a variety of software environments. XQL queries can be typed as strings on a command line, generated by graphical query interfaces or embedded strings in programs or even as a part of URL, in XML or HTML attributes. Apart from the way XQL queries are included and executed the following terms and operators are basic and required in order to extract the needed data:

TERMS & OPERATORS	**EXAMPLE**
Element name	computer
Wildcard as element name	*
Attribute name	@id
Wildcard as attribute name	@*
Equality	name='PK16'
Parent/child	computer/name
Ancestor/descendant	invoice//product
Subscripts	a[0] a[1,3-5,-1]

Filters	computer[name='CC-1']
Intersection	a[b] intersect a[0]
Union	a union b
Conjunction	a and b
Disjunction	a or b
Grouping	(a union b) intersect *[0]
Negation	not a

There are also under further study some additional operators to be added such as string containment, namespaces, comparisons other than equality, collection functions, return operators and sequence. In addition, Microsoft has included some new features which are optional and do not fit into the XQL evaluation model. These are:

- Methods;
- Ancestor(); and
- Id().

It is also important to note that the relationships among data contain a large proportion of the information contained in a document, which is one of the reasons that structured document formats like XML are useful in the first place. The following relationships are considered to be fundamental to the semantics of XML documents:

- **Hierarchy**
  - parent/child
  - ancestor/descendant
- **Sequence** (within a sibling list or in document order)
  - immediately precedes
  - precedes
- **Position** (within a sibling list or in document order)
  - absolute
  - relative
  - ranges

These relationships form the basic logical structure of XQL. They are also central to several other systems designed for addressing into SGML documents, including TEI Pointers, which have been used in academic circles for many years to index into documents, and XPointers, an addressing language being developed as part of the W3C XML activity.

XQL EXPRESSIONS

Most XQL expressions will evaluate either to a set of nodes or a Boolean value (true or false). All XQL query expressions may be said to evaluate to true or false. An expression evaluates to true if it is one of the following:

- A true Boolean value
- A set of Booleans containing at least one true value
- A non-empty set
- A single node (which evaluates to a non-empty set containing that node, and therefore to true)

An expression evaluates to false if it is one of the following:

- A false Boolean value
- A set of Boolean values containing no true values
- An empty set

XQL OPERATORS AND USAGE

Hierarchy between nodes is shown using the following operator	" / "
Wildcards **matches any element**, represents any element	" * "
Equal operator is used for comparisons of every kind, where it is needed	" = "
Using **attributes as identification** for elements	" @ "
Descendant operator, indicates any number of intervening levels	" // "
Filter operator filters the set of nodes to its left based on the conditions inside the brackets	" [ ] "
Return operators are analogous to the SELECT statement of SQL • Shallow returns: returns just the node to which it is applied • Deep returns: returns the element and all its children	Shallow returns ("?") Deep returns ("??")
Sequence operators allow the order in which data appears in a document to be used in query conditions • Precedes operator: evaluates its left-hand operand for the search context, then, for each node in the left-hand evaluation, evaluates to that node and all nodes that follow it. • Immediately precedes operator: evaluates to the set of node pairs in the search context in which the left operand immediately precedes the right operand	''precedes'' operator (;) - ''immediately precedes'' operator (;;).
• All Boolean operators are valid and used to construct complex queries	Or, and, not etc…

4.9 A Complete Example

The following is an invoice document. Traditionally, invoices are often stored in databases, but invoices are both documents and data. XQL is designed to work on both documents and data, provided they are represented via XML through some interface.

```
<?xml version="1.0"?>
<invoicecollection>
<invoice>
 <customer>
     Wile E. Coyote, Death Valley, CA
 </customer>
 <annotation>
      Customer asked that we guarantee return rights if these items
       should fail in desert conditions. This was approved by Marty
       Melliore, general manager.
 </annotation>
 <entries n=2>
   <entry quantity=2  total_price="134.00">
     <product maker="ACME" prod_name="screwdriver" price="80.00"/>
   </entry>
   <entry quantity=1  total_price="20.00">
     <product maker="ACME" prod_name="power wrench" price="20.00"/>
   </entry>
 </entries>
</invoice>
<invoice>
 <customer>
     Camp Mertz
 </customer>
 <entries n=2>
   <entry quantity=2  total_price="32.00">
     <product maker="BSA" prod_name="left-handed smoke shifter"
price="16.00"/>
   </entry>
```

```
    <entry quantity=1  total_price="13.00">
      <product maker="BSA" prod_name="snipe call" price="13.00"/>
    </entry>
  </entries>
</invoice>
</invoicecollection>
```

SAMPLE QUERIES

Suppose we wanted to see just the customers from the database. We could apply the following query:

Query:

```
  //customer
```

Result:

```
            <xql:result>
              <customer>
    Wile E. Coyote, Death Valley, CA
    </customer>
    <customer>
    Camp Mertz
    </customer>
            </xql:result>
```

We might want to look at all the products manufactured by BSA. The following query would manage it:

Query:

```
  //product[@maker='BSA']
```

Result:

```
  <xql:result>
    <product maker="BSA" prod_name="left-handed smoke shifter"
price="16.00"/>
    <product maker="BSA" prod_name="snipe call" price="13.00"/>
  </xql:result>
```

Filters are very useful when specifying conditions on paths that are not the same as what is returned. For instance, the following query returns the products ordered by Camp Mertz:

Query:

```
//invoice[customer='Wile E. Coyote, Death Valley, CA']//product
```

Result:

```
<xql:result>
  <product maker="ACME" prod_name="screwdriver" price="80.00"/>
  <product maker="ACME" prod_name="power wrench" price="20.00"/>
</xql:result>
```

4.10 Evaluation Of XQL

There are several opinions about how simple XQL is or how simple should it be. A lot of articles have been published across the web supporting several ideas.

People who claim XQL is too simple, complain about the lack of joins and transformations e.g. Microsoft's Adam Bosworth, in an article on the W3C Query Language Workshop by Lisa Rein , stated that "I think the XQL folks are trying to generalize path expressions to be a full query language, and I think this is a mistake. Query languages need other constructs than those that describe interesting elements to process. They need to say what to do with them (e.g. order them, extract important elements from them, sum them,...). I'm a huge fan of rich path expressions. I don't consider them a query language, just a useful part of one".

On the other hand, some people criticize XQL of being too complex e.g. Marcelo of Cantos, who is associated with the Structured Information Manager, has said that XQL "offers a good compromise between expressivity and simplicity" (XML Dev List, Fri, 26 May 1999).

Of course one is for sure. The set of features which are included in a query language is a matter of cost/benefit decisions. The more features it offers, the more complex it is. It seems to be a matter of personal opinion and evaluation. After all, we should not forget that XQL language is still under development and all kind of opinions are useful in order to achieve the best result!

4.11 Current Implementations Of XQL - Software

There are several current implementations of XQL. Some of them are presented here (in alphabetical order):

- **Datachannel:** "DataChannel RIO is an XML-enabled solution designed to build a dynamic two-way corporate portal with input (i.e. publishing) and output (i.e. retrieval) capabilities that make Intranets and Extranets easier to use, integrate, manage, and support."

- **GMD:** The GMD-IPSI XQL engine is a Java based storage and query application for large XML documents. The functionality may be accessed via command line invocation or the Java API. The engine consists of two main parts: (1) A persistent implementation of the W3C-DOM, and (2) A full implementation of the XQL language. The persistent DOM implements the W3C-DOM interfaces on indexed, binary XML files.

- **Microsoft Internet Explorer 5.0**: Microsoft's Internet Explorer 5.0 browser offers "extensive support for the latest standards-based web technologies, including Dynamic HTML, CSS, CSS-P, XML, XSL, XQL, and the W3C DOM"

- **ObjectStore:** ObjectStore's eXcelon "is a high-performance, highly scalable data server that caches and serves all information to enterprise applications and Web servers as XML. eXcelon can be used as an application cache for existing data sources, or as a complete data management system for new XML-based applications."

- **Software AG:** Software AG's Tamino "is the first information server to store XML information without converting it into other data structures. Tamino delivers XML information with exceptional performance for transaction-oriented applications within enterprises or on the Web. Tamino can also integrate data from existing databases into XML structures."

- **WebMethods:** webMethod's B2B "facilitates inter-enterprise integration between ERP applications, Web sites and legacy data sources. webMethods B2B 2.1 offers guaranteed delivery and enhanced continuous operation capabilities to support high-volume, mission-critical applications."

- **XML::XQL:** by Eduard Derksen (Enno), "is a Perl extension that allows you to perform XQL queries on XML object trees.

- **Xtract:** "Xtract is a command-line tool for searching XML documents. Just as `grep' returns lines which match your regular expression, so Xtract returns all those sub-trees from XML documents which match a query pattern. The query expression language is simple but powerful, and is based loosely on XQL, the recently proposed XML Query Language. An introduction to the Xtract query pattern language, together with the full Xtract grammar is in this tutorial." Although the author says it is "loosely" based on XQL, the discrepancy is slight: "The major difference from XQL is that a query must return a sequence of XML contents (either elements or text inside an element): it cannot for instance return just an attribute value.

5. SEARCHING

5.1 XML And Searching Across The Internet

The Internet's potential as a major revenue earner can only be realised if users can find exactly what they want on the Internet quickly, accurately, and with little effort. The development of tools which support the finding of relevant material within a few mouse clicks and key strokes is becoming increasingly critical, given the unprecedented rate at which the Web is growing and the rising numbers of novice users.

XML is said to be the language that will help the Internet user to find whatever he wants fast, accurately and easy.

Nowadays, even though internet searching is considered to have reach a high level, there are still some problems remaining that are related to the way searching is performed. To be more specific, most search tools index and search documents, which have no clear structure, such as text, HTML, word processor or even PDF files. These engines are mentioned as text search engines. Most of today's web pages available across the Internet are written in HTML. Although HTML is enough to include the necessary information for viewing text, it doesn't give us a way to describe the contents of the text: the meaning, the information *about* the text is lost because there is no way to store it or, in other words, to tag it. Furthermore, text searching applies to documents as whole and returns just a list of documents with some information as a result, and has no capabilities of combining data from multiple choices, which is a basic characteristic of query languages that XML searching is based on.

5.2 Searching And XML Metadata

An XML document is a structured document that consists of tags and data that are included into the tags. The meaning of the tag is to define precisely what kind of data are presented. For example, an <author> tag tells us that the data within the tag will be an author name (<author> Author's name </author>). This kind of information is called (structural) metadata. Metadata are very useful for searching because they provide us with the necessary information about the presented information. So, document metadata tells us about the author of the document, the date or the topic.

As a result of all the above, an XML document is a poor language if the goal is to simply represent information objects, but is more meaningful with respect to the information objects represented by text. The mark-up itself is a form of metadata, explaining to the user what the constituent elements are (by name) and how these information objects are structured into larger coherent units.

The presence of a query language that extracts the needed data from the XML documents accurately, fast and easy seems extremely important. XQL is XML's query language which has been implemented in order to satisfy the above mentioned goals. Without XQL development, talking about searching of XML documents would be useless as it would be impossible to achieve the locating of the necessary data.

5.3 Working Towards A Search-enabling XML Document Structure

There have been several XML documents structures proposed by people who are involved with XML, but it is unclear how well XML fits in with these.

Here is presented one of those, just as an example:

TITLE: An XML document should include a <TITLE> tag that can be used from search engines for displaying, after the search has completed.

META Keywords tag: Additional searchable keywords for this document.

META Description tag: A summary of the page for searching and displaying in a results list

META Robots tag: This tag was introduced to allow page creators to control whether robots would follow links on the page and whether search indexers would store words from this page. XML documents may want to use these as well.

LINKS: Search indexing robots traditionally follow links in HTML documents.

5.4 Problems That Might Appear

XML language is not widely known or used so there are some steps that should be made in order to get full advantage of it and use it properly.

- First of all, XML must be displayed in all of the web browsers that are available, and the structure standard should be just one. All users, or at least users that belong to specific groups, should have same structures for their various documents;
- Web site content providers will have to tag the pages according to the standard structures;
- As most of the web pages available today are written in HTML, it will be difficult if not hopeless to change them all. As a result, many of HTML pages that contain useful data might never be touched again;
- Search engines should be conformed with the new technology. So, search engine indexing applications will have to hold the tag information as metadata. In addition, the indexes should store the entire hierarchy, so that a searcher can specify the right tag even when the tag names are re-used in a structure. Furthermore, search engines will have to learn each standard structure in the collection of indexed documents and understand the DTD or other schema of the documents in the indexed collection;
- The interfaces of the search engines should represent the structural information, so that this can be available to users; and

- Searching for fields rather than the entire document is hard to be understood by a simple user. Users should adapt to the structure of documents in order to make their searches more effective. Search interfaces could use both types of searching, providing capabilities for text search or querying of XML documents.

As it is obvious, there are many problems which come up in case XML is used for searching Internet documents. These problems should be taken under serious consideration and solutions to them should also be found as soon as possible, in order to take full advantage of the new Internet searching technology.

5.5 Search Engines Based On XML

1) XML Text Search Engines

- XML Query Engine - JavaBean component for full-text indexing and searching of XML documents. Still in alpha phase. (http://www.fatdog.com/)

- Xdex (XMLIndex from Sequoia) - It was released in March, 2000. It promises searchable hierarchies, and a spider which can aggregate mark-up content from many sources. (http://www.xmlindex.com/)

- InfoGlide XML Similarity Search Engine – An unusual approach to finding useful documents even when the query and the data do not match exactly. (http://www.infoglide.com/technology/techprod.htm)

- Ultraseek version 3 supports searching XML files as full text, and setting up multiple sets of fields for field-specific searching.

- SIM (The Structured Information Manager) uses XML and full-text indexing to provide a powerful combination of database searching and text retrieval. (http://www.simdb.com/)

- The BUS (Bottom Up Scheme) engine is written in Java.

- GoXML Search Engine - does XML-specific search by providing a second step for the query with a popup menu of "context": the mark-up tag for the text. There does not seem to be a way to perform free-text searching as well, but it does seem to index by using a robot and spidering external sites. (http://www.goxml.com/)

2) XML Structured Query Engines

- fxgrep - parses an XML document for complex queries and finds all matches. Distributed as source code. (http://www.informatik.uni-trier.de/Eneumann/Fxgrep)

- WebMethods B2B QueryView - simple Java-based XQL engine example application, allows searching in a single document. (http://xml.webmethods.com/products/QueryView/)

- GMD-IPSI XQL Engine - Java-based persistent storage of XML documents in DOM format, XQL queries into the data store. Free for non-commercial use. (http://xml.darmstadt.gmd.de/xql/) See Globit Infonyte (http://www.globit.com/infonyte.htm) for the commercial version.

- XRS - XML Retrieval System is based on BUS and programmed in Java. It's very much concerned with document structure and searching on fields within documents. (http://dlb2.nlm.nih.gov/Edwshin/xrs.html)

- XSet is an XML database and high performance search engine library with a very simple tag-oriented query language, expressed in XML itself. (http://www.cs.berkeley.edu/Eravenben/xset/)

- UC Berkeley's Cheshire system indexes and searches structured data including XML, SGML and MARC records, as well as full-text data files. (http://cheshire.lib.berkeley.edu/)

- Xtenint information routing and selection agents perform real-time filtering of data streams using various pre-defined rules. (http://www.xtenit.com/)

- RDF for XML application from IBM: an application to build, manipulate and search RDF structures. It also converts RDF to and from XML. Written in Java, it requires the IBM XML for Java parser. (http://www.alphaworks.ibm.com/formula/rdfxml)
- The DataWare Knowledge Management system has announced support for XML. (http://www.dataware.com/)

- Lore - a database management system for XML using *semi-structured* data, with a special query language called Lorel. This is a research project in the Stanford University computer science department. (http://www-db.stanford.edu/lore/)

- YAT - a query language and data conversion development tool. (http://www-rocq.inria.fr/verso/Jerome.Simeon/YAT/)

- sgrep – a search tool that provides ‘grep’ searching for structured documents, including XML. This is a command-line program that lets you limit searches to contents of specified XML tags, such as "subject" or "evaluation". As of January, 2000, it has been updated to version 1.9 and now supports searching in hierarchies. (http://www.cs.helsinki.fi/~jjaakkol/sgrep.html)

5.6 CASE STUDY :The Goxml Search Engine: An Implementation Based On XML

PREREQUISITES:

A person using the GoXML Search Engine should already have understood the differences between XML and HTML. XML does not provide any mark-up semantics for telling a browser how to display a page. Instead, it allows authors of documents to mark up their data in a way that makes the data more useful to an end user.

A browser that supports XML, such as Microsoft Internet Explorer 5.0+, is necessary for using the GoXML Search Engine.

1) Viewing the following mentioned pages we can note differences between pages:

These pages have the person "Danny Sullivan" (he is the search engine Guru @ searchenginewatch.com) marked up in a variety of ways using XML.

a) In this document, he is marked up as Danny Sullivan, the WEBMASTER

- http://www.walkaboutwebs.com/SE/index.xml

b) In this document, he is marked up as a PLUMBER

- http://www.walkaboutwebs.com/SE/plumber.xml

c) Now he is marked up is a PROGRAMMER

- http://www.walkaboutwebs.com/SE/programmer.xml

d) Danny Sullivan appears to have another occupation as a BUILDER

- http://www.walkaboutwebs.com/SE/builder.xml

2) How to Search for Danny Sullivan

The XML mark-up has now supplied context to the preceding XML documents. Now we are going to search for Danny Sullivan. We open the Main search page and enter the term Danny Sullivan. Then we hit the "Ask GoXML" button

3) We can now supply a context to our results

The Search Results page that popped up looks exactly like any other search page. It contains a list of results for our search term "Danny Sullivan".

This is done by selecting a word that appears in a column to the left-hand side of the page. This list is collected by gathering all the XML tags in the documents that appear in the result list on the right hand side of the page. It is limited to words that the authors have used to mark up our keyword (in this case "Danny Sullivan").

RESULTS: If we click on the word "person" from the list on the left, GoXML will refine our search results to only those pages that contain the term "Danny Sullivan" within the context of "PERSON".

All other pages will vanish from the search results.

Let's try selecting these four different terms (from the list of words on the left hand side):

PROGRAMMER
BUILDER
WEBMASTER
PLUMBER

RESULTS: Each time we look for "Danny Sullivan" in a different context (the above words), we should get a completely different set of results. It should only find our search term Danny Sullivan within the context we specified.

ANALYZING THE FEATURES PROVIDED WHILE SEARCHING WITH THE GOXML SEARCH ENGINE

- **Tag Proximity**

<person>
<name>
<first> Tim</first >
<last> Bray</last >
</name>
</person>

If we use GoXML to search for Tim, in the context of <person >, we would get a positive match with a relevancy score. If we were to search for Tim, in the context of <name>, we would also get a positive match however, the relevancy will be higher based on the descriptor (name) being closer in the XML tree to the actual content "Tim".

The best result would be for a search for Tim, the "first name" as this carries the two closest context words to the actual Subject "Tim".

- **Case Sensitivity**

All XML tags are automatically converted to "upper" case when GoXML Spiders an XML site or a search query is matched. GoXML is therefore not case sensitive.

XML is however, case sensitive. XML Authors are reminded that the opening and closing tags *MUST* be matching as far as case goes, otherwise, their documents are not considered valid.

eXtensible Object Server Pages (*.XOSP) conform to the syntax of XML and require proper case to be used.

- **XML Content**

If we are searching for content, it is important to understand how GoXML looks for content within XML tags. Consider using this example:

<person>
<name>
<first>Tim</first>
<last>Bray</last>
</name>
</person>

If we search for "a person named Tim", our search query will be directed to all web sites which have the <person> tag. The query mechanism will then look to see if the word "Tim" is present within the tag. This is important to authors as it gives us a better understanding of the importance of the root (first) tag in an XML document.

A search for the content "Tim" will be positive if we search in the context of "person", "name" or "first", however, if we search in the context of "last" our search result will be negative.

Likewise, a search for "name" in the context of "person" will also not be successful, as anything within the angle brackets name, is not deemed to be actual content.

- **XML Attributes**

An XML document may contain many attributes. Modifying the previous example to:

< person>
<name type="first" >Tim </name>
<name type="last" >Bray </name>
</person>

attributes are converted to the same index value as an actual tag. The XML index does not store attribute name=value pairs as unique content. This new example would behave much the same way as the previous example with the following exceptions:

A search for "Tim" within the context of person and name will still be successful.
A search for "Tim" within the context of "last name" will not be successful.

- **XML Namespaces**

XML Namespaces presented us with a unique challenge. The development team of the GoXML considered best to leave the namespace as a complete tag. The following example shows what they support:

<RPM:Name> gwydion-dylan</ RPM:Name>
<RPM:Version> 2.1</RPM:Version >
<RPM:Release> 19980916_libc6</ RPM:Release>
<RPM:URL> http://www.randomhacks.com/dylan</ RPM:URL>

This would be saved as:

RPM:NAME:gwydion-dylan
RPM:VERSION:2.1
RPM:RELEASE:19980916_libc6 etc.

The reasoning behind this decision is that GoXML can provide us with a list of the tags, complete with namespace prefixes, if we do not specify a context for our search.

- **XML Meta Tags**

There is currently no standard for which we can index XML meta tags. We have accomplished some work on a standard for XML meta tags. The main stumbling block against it's adoption is the issue of standards. No proposal will be submitted to the W3C .org until such time as there is further participation by industry groups.

Using comment tags had been considered (see below). It was dropped in August of 1999. It is an opinion of the team which developed the GoXML Search Engine, along with other developers, that comments are parts of data and their presence is not required by ANY application which may use the data. Because meta tags are used by the GoXML Search Engine (the application), they do not fall into this category.

<!--XMETA:KEYWORDS | keyword1 keyword2 keyword3-->

<!--XMETA:DESCRIPTION | This describes a very bad way to implement indexing of XML Meta tags on GoXML.com. Comments are not to be used except for making comments.-->

It has been suggested on newsgroups that this would be placed at the top section of the XML document.

Another proposed syntax for META tags is through the use of processor instructions or PI's. Such a PI may look like:
<?META name="..." content="..."?>

- **Support for the Robots Exclusion Standard**

GoXML supports the robots exclusion standard. The Robots exclusion standard contains information for preventing your data from being spidered and indexed by Search Engines. It is expected that some XML data will be of sensitive information and therefore should not be indexed.

- **XML - DSO Binding supported**

GoXML.com can spider and index XML data bound to an HTML document using *DSO binding*.

DSO binding is a technique where XML data can be taken from a Server source and bound client-side to the web document. Microsoft® Internet Explorer 5 ships with a C++ Data Source Object (DSO) engine that can be used to bind XML to HTML. This replaces the Java based XML DSO that was supported in version 4.0. The new C++ DSO system is heavily favoured by many web developers using XML as it allows for the structure and intelligent nature of XML data while allowing the display semantics of HTML to be used. C++ DSO allows you to bind directly to an XML Island within HTML. Data Source Object (DSO) binding also gives authors an intermediate solution to the instability of XSL (eXtensible Style sheet Language).

- **XML Query Interface**

The GoXML Search Engine has also made XML Query Interface available to the public. This service allows submitting a search over TCP/IP in well-formed XML, and receiving the results back in well-formed XML.

The usage of the engine include remote queries and insertions that can be made to the search engine, and these don’t need to be cgi applications. (data can be retrieved from other applications).

Speaking to GoXML is performed by sending an XML document to it's port (or socket). The response will be an XML document. The server to use is: xqi.goxml.com on port 5910.

Example : Query database for "Apple" in "Tree"

```
<?xml version="1.0"?>
<GOXML>
 <QUERY>
  <KEYWORD>apple</KEYWORD>
  <TAG>tree</TAG>
 </QUERY>
</GOXML>
```

The query 'Look for an "apple" in a "tree"' will be performed, and the result will be in the form of:

```
<?xml version="1.0"?>
<QUERY TYPE="URL" HITS="1" TAG="TREE">
 <kEYWORDS>
  <WORD>APPLE</WORD>
   </KEYWORDS>
   <HIT ID="234">
    <WORD POSITION="0" WORDCOUNT="5">APPLE</WORD>
    <URL>http://www.eden.org/fruit/forbidden.xml</URL>
    <DESRIPTION>Forbidden fruit</DESCRIPTION>
    <SYNOPSIS>...</SYNOPSIS>
   </HIT>
 </QUERY>
```

6. XML AND DATABASES

6.1 Introduction

In this chapter, we are truing to examine the relationship between XML and Databases. We study how XML can be used as a means to transfer data between heterogeneous databases. Specifically, we present some techniques used to model relational data in XML documents, and how we can apply the XML APIs SAX and DOM directly to a database, instead of converting it first to a XML document. The repository requirements for XML data and documents is also examined, as well as the XML support in Oracle tools, and especially Jdeveloper and PL/SQL Utilities.

The most important factor in choosing a database is whether the database will be used to store *data* or *documents*. If data is being stored, then a database tuned for data storage, such as a relational or object-oriented database, is necessary as well as a middleware to transfer data between the database and XML documents. On the other hand, if documents are going to be stored, what is needed is a content management system designed specifically to store documents.

6.2 Categories of XML documents

XML documents fall into two rough categories: *data-centric* and *document-centric*. Data-Centric Documents are characterized by fairly regular structure, fine-grained data (i.e. the smallest independent unit of data is at the level of a PCDATA-only element or an attribute), and little or no mixed content. The order in which sibling elements and PCDATA occurs is often not significant. Data-centric documents are usually designed for machine consumption. Many prose-rich documents are, in fact, data-centric. Any Web site that dynamically constructs HTML documents today by filling a template with database data can probably be replaced by data-centric XML documents (containing page-specific text retrieved from the database) and one or more XSL style sheets.

Document-centric documents are characterized by irregular structure, larger grained data (i.e. the smallest independent unit of data might be at the level of an element with mixed content or the entire document itself), and lots of mixed content. The order in which sibling elements and PCDATA occurs is almost always significant. Document-centric documents are generally designed for human consumption. [BOUR99]

6.3 Mapping Document Structure to Database Strusture

In order to transfer data between an XML document and a database, it is necessary to map document structure to database structure and vice versa. Such mappings fall into two general categories: template-driven and model-driven.

6.3.1 Template-driven mapping

In Template-Driven mappings, there is no predefined mapping between document structure and database structure. Instead, commands such as SELECT statements are embedded in a XML template that is processed by the data transfer middleware. For example:

```
<FlightInfo>
 <Intro>The following flights have available seats:</Intro>
 <SelectStmt>SELECT Airline, FltNumber, Depart, Arrive FROM Flights</SelectStmt>
```

```
 <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```

When processed by the data-transfer middleware, each SELECT statement might be replaced by its results, formatted as XML:

```
<FlightInfo>
 <Intro>The following flights have available seats:</Intro>
   <Flights>
      <Row>
        <Airline>ACME</Airline>
        <FltNumber>123</FltNumber>
        <Depart>Dec 12, 1998 13:43</Depart>
        <Arrive>Dec 13, 1998 01:21</Arrive>
      </Row>
      ...
   </Flights>
 <Conclude>We hope one of these meets your needs</Conclude>
</FlightInfo>
```

Template-driven mappings can be very flexible, since it may allow to place result set values wherever in the result set including using them as parameters in a subsequent SELECT statement, or support parameterisation of SELECT statements, such as through HTTP parameters. Currently, template-driven mappings are available only for transferring data from a relational database to an XML document. [BOUR99]

6.3.2 Model-driven mapping

In a model-driven mapping, a data model of some sort is imposed on the structure of the XML document and this is mapped to the structures in the database and vice versa. What is lost in flexibility is gained in simplicity. XSL is commonly integrated into model-driven products to provide the flexibility found in template-driven systems.

Two models for viewing the data in an XML document are common. The first of these, used for transferring data between an XML document and a relational database, models the XML document as a single table or set of tables. The structure of the XML document must be similar to the following, where the `<database>` element does not exist in the single-table case:

```
   <database>
      <table>
         <row>
            <column1>...</column1>
            <column2>...</column2>
            ...
         </row>
         ...
      </table>
      ...
   </database>
```

The second common model for data in an XML document is a tree of objects, in which elements generally correspond to objects and attributes and PCDATA correspond to properties. This model maps directly to object-oriented and hierarchical databases and can be mapped to relational databases using traditional object-relational mapping techniques. This model is **not** the Document Object Model (DOM); the DOM models the document itself, not the data in the document. When modelling an XML document as a tree of objects, there

is no requirement that elements necessarily correspond to objects. Similarly, it is sometimes useful to model elements with mixed or element content as properties. [BOUR99]

6.4 Modeling Relational Data in XML

XML provides a compelling environment for the integration of disparate forms of data in a platform independent manner. There is a spectrum of potential approaches in mapping information from traditional (e.g. relational, object-oriented) data models to XML. At one end of the spectrum is the custom mapping of individual pieces of information from the database into a *"pre-ordained"* schema (e.g. an industry standard interchange format). This has the drawback of significant custom code to perform the mappings between the two loosely coupled schemas. At the other end of the spectrum is a *"data-dump"* of the entire contents of a database with the intention of re-constituting the database with all of its relationships and data intact. This approach tends to yield a mechanical mapping onto a generic metadata schema, such as XMI. It has the drawback of the obscuring of the business concepts, since the schema is about the metadata concepts themselves.

In this point, we are going to present an approach that lies between these two extremes and apply it to relational data bases. It provides mechanical mappings of key database structure concepts while retaining the notion that the schema should be about the business information. This approach has the merit of yielding straightforwardly understandable schemas which are in agreement with existing data structures.

A simple example to work is the following employee database with three tables. EMPLOYEE stores personal information and a summary of the term of the relationship. PERF_REVIEW stores performance reviews given to the employee. COMP_CHANGE stores changes to compensation resulting from a performance review.

```
TABLE EMPLOYEE (
      NUM         LONGINT PRIMARY KEY,
      FNAME       STRING(32),
      LNAME       STRING(32),
      HIRE_DATE   DATE,
      TERM_DATE   DATE MAY BE NULL );
TABLE PERF_REVIEW (
      EMP_NUM     LONGINT PRIMARY KEY FOREIGN KEY,
      REVIEW_DATE DATE PRIMARY KEY,
      REVIEW      TEXT );
TABLE COMP_CHANGE (
      EMP_NUM     LONGINT FOREIGN KEY,
      REVIEW_DATE DATE MAY BE NULL,
      EFF_DATE    DATE,
      SALARY      INT );
```

The DTD syntax is very efficient in expressing a certain set of constraints on conforming documents. However, it knows nothing about concepts such as data types and key relationships. So, DTD should be extended in order to capture such additional information for the Element Types and Attribute Types. We'll use fixed attributes to accomplish this because the value of fixed attributes is unchangeable in individual

instances of an element so they can be thought of as properties of the element type itself rather than of an instance. For our purposes we will need four such properties:

`dtype` : The datatype for the element or attribute type. This is used to more fully specify the allowed values for an element of type `PCDATA` or an attribute of type `CDATA`.

`dsize` : The storage size and/or precision for the datatype.

`pkey` : The name of the attribute or element type(s) which corresponds to the primary key column. In the case of aggregate keys, this is a space-delimited list. This is used only on the element type corresponding to a table.

`fkey` : A reference to the attribute or element type which represents the column to which this attribute or element type refers. This is used only on the attribute or element type corresponding to a column that contains a foreign key.

These metadata properties are also appropriate with minor modification to other schema syntaxes such as XML Data and SOX. Essentially, they become additional attributes on the element type or attribute type declaration.

While modelling relational data we can have an element named after the table that corresponds to a row of data with attributes containing the values for each of the columns, or we could equally have sub-elements containing the column data:

- **Modelling Tables**

1. The table itself becomes an element type. In the document each occurrence of an element of this type will contain a single row of the data. If we're modelling the columns of this table as attributes, the content model is `EMPTY`, if we're modelling the columns as elements, the content model is a sequence of the elements which correspond to the columns. In the later case we make optional (?) any elements whose columns' values may be null.

```
<!ELEMENT EMPLOYEE EMPTY>
```

or

```
<!ELEMENT EMPLOYEE (
   EMPLOYEE.NUM,
   EMPLOYEE.FNAME,
   EMPLOYEE.LNAME,
   EMPLOYEE.HIRE_DATE,
   EMPLOYEE.TERM_DATE?
   )>
```

2. If the table has a primary key we capture that information with the pkey metadata property.

```
<!ATTLIST EMPLOYEE e-pkey NMTOKEN #FIXED 'EMPLOYEE.NUM'>
```

3. To provide id/idref access to key relationships we create a special ID attribute to contain a globally unique version of this key for each element instance.

```
<!ATTLIST EMPLOYEE pkey_id ID #REQUIRED>
```

- **Modelling Columns as Elements**

  For each column:

  1. Column Name. A new element type with the same name as the column is created. In the document, each occurrence of an element of this type will contain the value of a single column of a single row of data. Because element type names must be unique throughout the whole document, you should probably qualify the column's element type name by prefixing it with the tables' name.

     ```
     <!ELEMENT EMPLOYEE.TERM_DATE (#PCDATA) >
     ```

  2. Column Data Type. Data types are captured using the dtype metadata property introduced in Extending the DTD.

     ```
     <!ATTLIST EMPLOYEE.TERM_DATE e-dtype NMTOKEN #FIXED 'date' >
     ```

  3. Column Foreign Key. If the column contains is a foreign key, we use the fkey metadata property to record the column's element to which the key refers.

     ```
     <!ATTLIST PERF.REVIEW_EMP_NUM e-fkey NMTOKEN #FIXED 'EMPLOYEE.NUM' >
     ```

  4. To provide id/idref access to key relationships, a special attribute of the table's element type may be created to serve as an IDREF link to the element corresponding to the other table. The attribute name is formed by appending "_idref" to the column element's name.

     ```
     <!ATTLIST PERF_REVIEW PERF_REVIEW.EMP_NUM_idref IDREF #REQUIRED >
     ```

- **Modelling Columns as Attributes**

  For each column:

  1. Column Name. A new attribute type with the same name is created. In the document, each occurrence of an attribute of this type will contain the value of a single column of a single row of data.

     ```
     <!ATTLIST EMPLOYEE EMPLOYEE.TERM_DATE CDATA #IMPLIED>
     ```

  2. Column Data Type. Data types are captured using the dtype metadata property.

     ```
     <!ATTLIST EMPLOYEE a-dtype NMTOKENS #FIXED
           'EMPLOYEE.NUM int
           EMPLOYEE.FNAME string
           EMPLOYEE.LNAME string
           EMPLOYEE.HIRE_DATE date
           EMPLOYEE.TERM_DATE date'
           a-dsize NMTOKENS #FIXED
           'EMPLOYEE.FNAME 32
           EMPLOYEE.LNAME 32'>
     ```

  3. Column Nullable. If the column may contain a null value then the attribute type is made implied, if not, it is made required.

4. Column Foreign Key. If the column contains is a foreign key, we use the fkey metadata property to record the table element and attribute to which the key refers.

   ```
   <!ATTLIST PERF_REVIEW a-fkey NMTOKENS #FIXED EMP_NUM EMPLOYEE.NUM'>
   ```

5. To provide id/idref access to key relationships, a special attribute of the table's element type may be created to serve as an IDREF link to the element corresponding to the other table. The attribute name is formed by appending "_idref" to the column element's name.

   ```
   <!ATTLIST PERF_REVIEW EMP_NUM_idref IDREF #REQUIRED >
   ```

By adopting some simple conventions, XML Schemas can successfully model information sources such as relational databases. By capturing datatype, key relationships and id/idref information, extracted data can retain the metadata needed to facilitate processing at both side of information exchange. [BUCK99]

6.5 XML APIs for Databases

Databases and XML offer complementary functionality for storing data. Databases store data for efficient retrieval, whereas XML offers an easy information exchange that enables interoperability between applications. In order to take advantage of XML's features database tables can be converted into XML documents. XML tools can then be used with such documents for further processing. For example, XML documents can be presented as HTML pages with XSLT style sheets, can be searched with XML-based query languages such as XQL, can be used as a data-exchange format, and so on. However, converting a database into an XML document is an expensive approach, one that requires not only the initial cost of *conversion* but also the subsequent costs of *synchronizing* both information sources. For processing XML documents, most XML tools work with the SAX or DOM API. There is a way, yet, to implement the same APIs directly over a database, enabling XML tools to treat databases as if they were XML documents and obviate the need of converting a database.

- **The basics**

Because of a database's highly regular data-storage structure, it can be mapped into data-centric XML documents. For example, a database table can be transformed into an XML document with a DTD of the following form:

```
<!ELEMENT table (rows)*>
<!ELEMENT rows (column1, column2, ..., columnN)>
<!ELEMENT column1 #PCDATA>
<!ELEMENT column2 #PCDATA>
....
<!ELEMENT columnN #PCDATA>
```

It is necessary, additionally, to declare every table of the database in the DTD of the XML document, since each table may have a different number of columns.

With an XML API for databases, one can make the database *look* like an XML document; these APIs present the database as a virtual XML document. The tools using such an XML API need not care whether they are operating on a database table or an XML file.

- **Implementing the SAX API for Databases**

To implement the SAX API for Databases, it is necessary to implement a parser that operates on a JDBC data source, iterates over each row and column, and generates appropriate SAX events while iterating. It might be useful also to use XPointer/XLink in order to set the reference to a foreign key in a table.

- **Implementing the DOM API for Databases**

To build a DOM tree for a database table, a class should be written which would iterate over rows and columns and build nodes for a tree as it visits them. Another approach simpler and requiring less coding could reuse an existing library, which builds a DOM tree from a SAX event stream.. To implement the DOM API using such an approach it needs just a clever reuse of the SAX parser for databases.

- **Using the SAX API for Databases**

Except for using the SAX API for Databases to implement a DOM API for Databases, it could be also used to integrate with XT (an XSLT processor written in Java). With such integration, an XSLT style sheet can be applied directly to the virtual XML documents stored in a database. This approach can be used to generate HTML pages directly from a database using an XSLT Style sheet without incurring the penalty of creating an intermediate XML file. The style sheet formats a highly regular XML document represented by a database table, which is formatted as an HTML table. The style sheet uses names of markers for columns as table headers. Moreover, the integration of the SAX API for databases and XT could be used to convert a database into a nonvirtual XML document.

- **Using the DOM API for Databases**

For most situations, the SAX API for Databases is more memory efficient than the DOM API for Databases. However, some applications need random access to the XML documents and therefore require the tree-like structure that the DOM API for Databases offers. The DOM API for Databases could be integrated with an XQL processor. XQL, the query language for XML documents, has a syntax similar that of to XPath patterns. By integrating the DOM parser for database with GMD-IPSI's XQL Engine, SQL-like queries could be performed on the XML document representing a database table. [LADD2000]

6.6 Storing and Retrieving XML Documents

The term *content management system*, as opposed to *document management system*, reflects the fact that such systems generally allow to break documents into discrete content fragments, as well as metadata, rather than having to manage each document as a whole. This does not only simplify such things as coordinating the work of multiple writers working on the same document, but also allows to assemble entirely new documents from existing components.

Content management systems generally support "round-tripping" of documents, as things such as physical structure are often critical to document maintenance. Content management systems also provide a number of other capabilities, including:

- Version and access control;
- Search engines;
- Editors;
- Publishing engines, such as to paper, CD, or the Web;
- Separation of content and style;
- Extensibility through scripting or programming; and

- Integration of database data

Relational databases do not inherently deal well with many of the things needed by a content management system: order, hierarchy, irregular structure, and fields of highly variable length. So, most content management systems are built on object-oriented or hierarchical databases.

The simplest way to store XML documents in a relational database is to store each document in unparsed form in a single column, which has obvious drawbacks, such as the inability to create a new document from parts of existing documents. However, relational databases are steadily increasing their text processing capabilities, with full-text indexing and the ability to perform specialized searches, such as proximity searches or searches that use thesauri to look for synonyms. [BOUR99]

6.7 XML Repository Requirements

XML storage requirements are very different from those of its predecessor, HTML. XML extends HTML's simple unidirectional linking, adding support for links to multiple targets, indirect addressing and bidirectionality. Handling this rich linking requires a storage mechanism with far more powerful management of references between objects than that provided by the file system or relational databases. The ideal XML repository should handle:

- large data volumes (larger than physical memory);
- concurrent editing;
- link management and navigation;
- full-text queries;
- standardized API access;
- BLOB (Binary Large OBject) management;
- revision control;
- legacy entity management;
- metadata with associations to legacy data types; and
- hierarchical namespaces.

The architectures of relational databases and file systems do not map well to the richly interlinking hierarchical architecture of structured XML content. Only object databases can effectively store, manage and manipulate XML data. A comparison between the most popular types of storage follows:

- **File System Storage of XML Data**

XML applications must store and index the fine-grained elements as well as the document structure. In addition, they must be capable of linking these fine-grained elements directly to each other and to a variety of data types containing associated information. Attempts to parse XML data of any complexity would overwhelm the capabilities of the file system to maintain the rich linking structure and semantics of the data.

- **Relational Database Storage of XML Data**

Their table-based data model is very poorly suited to the hierarchical, interconnected nature of structured XML content. RDBMS are further hampered by the fact that they must represent the tree structure of XML content with an inefficient set of tables and joins. The XML object's structure and semantics are either lost, minimizing its value, or they must be duplicated in the design of the database. Since relational databases

decompose XML elements into various tables, linked via keys, it is very difficult to implement an effective locking scheme that doesn't dramatically hinder concurrent use and scalability.

- **Object Database Storage of XML Data**

Object-DBMS is ideal for the creation and management of hierarchical XML trees, while providing both hierarchical tree navigation and rich link traversal. The rich relationship linking of an object-oriented model enables both hierarchical navigation and rapid branch traversal, reducing computation and increasing performance. Object databases are also designed to handle arbitrary, variable-length data types and interrelated data. Object databases allow also object-level locking, providing for a much more granular locking than relational or file system-based solutions. [HOGA97]

Evaluating the requirements of applications in this field, the following criteria for the selection of a repository are critical:

- Scalability;
- language support;
- ease of programming; and
- embeddability.

Summarizing, the ideal repository for XML data would offer:

- tightly integrated XML-specific tree navigation;
- versioning;
- management of arbitrary links;
- import/export;
- publishing of structured content on the web; and
- support for object-oriented programming languages as well as common scripting languages etc.

6.8 XML Support in Oracle Tools

Most critical business data is managed by relational databases. In order to realize the promise of XML as an enabler for exchanging business information, we must be able to read and write XML data to and from the database and to integrate this data with existing applications.

Oracle's XML products and technologies allow applications to parse XML data and store it in the database and build XML documents dynamically out of the database. The Oracle XML Developer's Kit (XDK) contains the basic building blocks for reading, manipulating, transforming and viewing XML documents. To provide a broad variety of deployment options, the Oracle XDK is available for Java, C, C++ and PL/SQL. Unlike many shareware and trial XML components, the Oracle XDK is fully supported and comes with a commercial redistribution license. The Oracle XDK consists of the following items:

- XML Parsers: supporting Java, C, C++ and PL/SQL, the components create and parse XML using industry standard DOM and SAX interfaces;
- XSL Processor: transforms or renders XML into other text-based formats such as HTML;

- XML Schema Processor: allows use of XML simple and complex data types;
- XML Class Generator: automatically generates Java and C++ classes to send XML data from Web forms or applications;
- XML Transviewer Java Beans: visually view and transform XML documents and data via Java components; and
- XSQL Servlet: combines XML, SQL, and XSLT in the server to deliver dynamic web content.

With the integration of XML technology, the Oracle Internet Platform, which includes Oracle8i, Oracle Application Server and new message broker technologies, is an increasingly powerful alternative for deploying Internet applications.

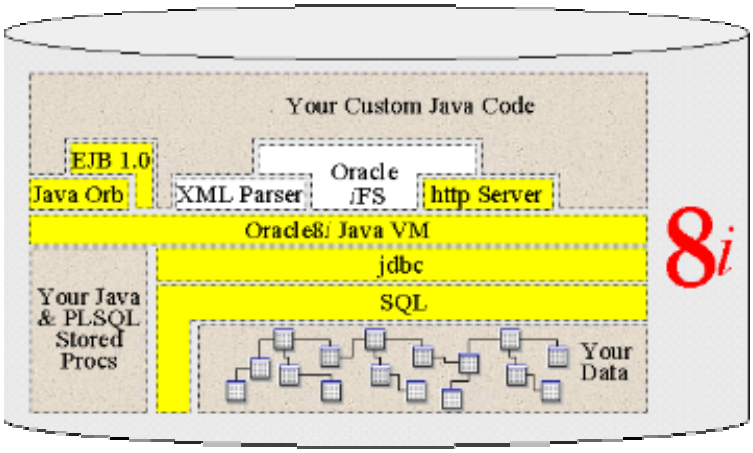


Figure : Oracle8*i* Platform for Java/XML

Running on top of the Oracle8*i* Java VM are a set of platform services supporting industry-standard interfaces for developers to build robust, database-driven applications in Java and PL/SQL. These services include:

- Java Stored Procedures for integrating Java code with SQL and PL/SQL;
- Object Request Broker based on VisiBroker for Java for distributed object communications;
- Enterprise JavaBeans 1.0 Server for server-based transactional components;
- Web Server to enable invoking Java Servlets running in the database from a browser over HTTP; and
- Internet File System (*i*FS) for organizing and accessing documents and data using a file/folder-based metaphor through standard Windows & Internet protocols. [ORAC1]

6.8.1 XML Support in JDeveloper 3.1

Oracle JDeveloper provides a complete Java development environment for developing and deploying applications ranging from Java and HTML clients to server-based business components for the Internet computing platform. JDeveloper with Oracle Business Components for Java is a full-featured application development tool that offers integrated support for building end-to-end e-business applications for the

Internet. JDeveloper offers an integrated environment for application developers to productively develop, debug, deploy, reuse, and customize multi-tier, component-based Java and XML applications.

The same application component can be deployed as JavaServer Pages (JSP), Java servlets, Enterprise JavaBeans, or CORBA components without modification to the application code. This deployment flexibility allows companies to deploy the same application onto a variety of deployment servers and to distribute the same application to a variety of clients including Web browsers, hand-held and wireless devices.

JDeveloper uses XML internally and enables development of XML applications. The Business Components for Java framework that ships with JDeveloper uses XML to store metadata about its application components. Important information is now stored in a structured document rather than in Java source code. This makes the application easier to understand and more importantly easier to customize.

JDeveloper 3.1 offers many features that enable developers to create business-to-consumer and business-to-business XML applications. JDeveloper can be utilized to write XML documents and XSL Style sheets, to generate XML on the fly using the Business Components for Java Framework or the database directly and to transform XML into HTML, WML, XML or any other format.

JDeveloper 3.1 integrates the various XML Utilities that Oracle has produced, including the Oracle XML Parser for Java, the XML SQL Utility and the XSQL Servlet. JDeveloper provides an integrated development environment for developers by offering editors, compilers and (remote) debuggers for Java and XML.

The XSQL Servlet offers a productive and easy way to get XML in and out of the database. Using simple scripts developers can:

- Generate simple and complex XML documents;
- Apply XSL Style sheets to generate into any text format;
- Parse XML documents and store the data in the database; and
- Create complete dynamic web applications without programming a single line of code.

Portal To Go and Oracle JDeveloper together offer an extremely powerful environment for developing mobile applications. Developers can use JDeveloper to generate XML from the database or from a Business Components for Java Application and use Portal To Go to deliver content to Web browsers, PDAs or Cell phones. From the development environment developers can now test and debug the total application. JDeveloper provides remote debugging to test the mobile application when deployed. [ORAC2]

6.8.2 XML Utilities for PL/SQL in Oracle

- **DBXML :** Format SQL query results as XML
- **DBDOM :** The most common and useful parts of the DOM API in PL/SQL
- **DBXSL :** XSL transformations in PLSQL

DBXML

DBXML is a PL/SQL package that is useful for formatting the results of a SQL query as a well-formed XML document. Future releases of Oracle8i will include native support for formatting query results in XML.

DBXML has two interfaces, one of which offers more control (**DBXML.Get**) and the other offers more simplicity (**DBXML.Query**). DBXML.Query invokes DBXML.Get internally, but offers a simpler API for the most common cases. It can take several parameters, the only one of which is required is **theQuery,** that is **t**he SQL statement we want to execute

For more control over the XML result, the lower-level API that DBXML provides is the DBXML.Get procedure. It can also take certain parameters. The only one that is required is **tb,** the table name we want to query.

If Oracle Web Agent is set up correctly, DBXML.Query or DBXML.Get can be invoked directly in an URL using the syntax:

http://yourmachine/youragent/package.procedure?param1=value1&param2=value2

For example:

http://yourmachine/xmldemo/owa/dbxml.query?theQuery=select * from emp where ename='JONES'

DBDOM

Oracle 8i includes the Oracle XML Parser, implemented in Java, inside the database. DBDOM is an implementation of the most common and useful parts of the Document Object Model Level1 Core API in PL/SQL, plus a few helper functions.

The DBDOM package can be used to:

- Programmatically create well-formed XML documents in memory;
- Write XML document to file, string buffer, or to HTTP stream;
- Read and Parse XML document from file, string buffer, or HTTP stream; and
- Search an XML document for a simple XSL Search pattern

Behaviour of DBDOM functionality on documents which are not Well-Formed in the strict XML sense is not predictable and effectively undefined. DBDOM's XML parser is neither a well-formedness parser nor a validating parser. It is simply a string tokenizer which is aware of XML Elements, Processing Instructions, Comments, and Text Nodes. CDATA sections are not yet supported in this prototype and support for Attributes is usable but not complete.

DBDOM defines the following types:

- DOMDocument
- DOMNode
- DOMNodeList

And the following constants to represent the node types it is capable of working with.

- NODE_TEXT
- NODE_PROCESSING_INSTRUCTION
- NODE_ELEMENT
- NODE_COMMENT

DBXSL

DBXSL is a prototype package which generates style sheets in order to produce a visual representation of data as an HTML "tree" in the browser. There are two different XSL transformations which are combined in series. The first XSL transformation ("Tr-1") transforms the tree of DBXML data into a canonical representation of a <TREE> of <NODE> nodes. The second transformation ("Tr-2") transforms the canonical tree of nodes into an HTML "tree".

The "Tr-1" transformation is different depending on the table. The "Tr-2" transformation is the same for any information that can be represented in the canonical tree of nodes, meaning it can be static. [ORAC3]

7. WEBDAV

7.1 Introduction in WebDAV (HTTP Extension)

Nowadays, that the worldwide use of the Internet is a reality, a pressing need occurs in the corporate world: Large companies with physically dispersed divisions create distributed teams to work together on software projects. Cross-organizational projects also occur with greater frequency, such as a subcontractor working closely with a primary systems-integration contractor on a large project.

These geographically dispersed teams share the same needs for distributed source-code control. When it comes to working on the design documents, test cases, specifications, and source code that comprise the project, individual team members need to work on pieces in isolation, then integrate those pieces with the modifications of their coworkers, without clobbering anyone else's changes. Changes need to be tracked so that errors and exploratory design changes can be undone easily. Tracking creates a group memory of how files have changed over time, which is valuable for later reconstruction of detailed design rationales. Released and stable configurations of the project are tracked so they can be regenerated quickly, and so that bug fixes can be made to the appropriate release.

CVS(Concurrent versions system) was just the beginning. But this wasn't enough because the needs were still greater. HTTP protocol had to be somehow extended in order to cover the needs. WebDAV promises to make it easier to perform remote collaborative project work over the web. It extends the web to make authoring of web resources as easy as browsing them.

To be more specific, as the Working Group chairman Jim Whitehead (University of California, Irvine) claimed: "WebDAV (Web-based Distributed Authoring and Versioning) is an Internet Engineering Task Force (IETF) working group that seeks to extend HTTP 1.1 for distributed authoring and versioning. It uses XML".

In other words, Webdav is a set of extensions to the HTTP/1.1 protocol which allows users to collaboratively edit and manage files on remote web servers. It simply adds new HTTP methods and headers and in addition, it specifies how to use the new extensions, how to format request and response bodies or how existing HTTP behaviour may change. It is an IETF proposed standard (published as RFC2518), and was approved in December 1998.

7.2 Goals Of The WebDAV

The stated goal of the WebDAV working group is (from the charter) to "define the HTTP extensions necessary to enable distributed web authoring tools to be broadly interoperable, while supporting user needs", and in this respect DAV is completing the original vision of the Web as a writable, collaborative medium.

However, WebDAV also has some more extended goals, which stand beyond simple web authoring. Some view DAV as a network filesystem suitable for the Internet, one that works on entire files at a time, with good performance in high-latency environments. Others view DAV as a protocol for manipulating the contents of a document management system via the Web. An important goal of DAV is to support virtual enterprises, being the primary protocol supporting a wide range of collaborative applications. Importantly, a major goal is the support of remote software development teams. A final goal of DAV is to leverage the success of HTTP

in being a standard access layer for a wide range of storage repositories -- HTTP gave them read access, while DAV gives them write access.

All of these goals are complementary, and are satisfied by the same network protocol.

The WebDAV working group "found that there was a pressing need to develop standard extensions to the HyperText Transfer Protocol (HTTP) for the following capabilities:

1) **Metadata.** The ability to create, remove, and query information about Web pages, such as its author, creation date, etc. Also, the ability to link pages of any media type to related pages;
2) **Name space management.** The ability to copy and move Web pages, and to receive a listing of pages at a particular hierarchy level (like a directory listing in a file system);
3) **Overwrite prevention.** The ability to keep more than one person from working on a document at the same time. This prevents the "lost update problem" in which modifications are lost as first one author, then another writes their changes without merging the other author's changes; and
4) **Version management.** The ability to store important revisions of a document for later retrieval. Version management can also support collaboration by allowing two or more authors to work on the same document in parallel tracks." [from the Introduction of the working draft]

Finally, the basic idea on which WebDAV was based on the following:

"The DAV property model is based on name/value doubles. The name of a property identifies the property's syntax and semantics, and provides an address with which to refer to a property. The name and value of a property is expressed as a well-formed XML element, where the name of the property is the name of the XML element, and the value of the property must be either blank, or a well-formed XML element value."

7.3 Relation Between XML And WebDAV

The relation between XML and WebDAV is obvious from the following extract taken from the first Internet draft of IETF:

"Unlike HTTP/1.1, WebDAV encodes method parameter information either in an Extensible Markup Language (XML) request entity body, or in an HTTP header. The use of XML to encode method parameters was motivated by the ability to add extra XML elements to existing structures, providing extensibility; and by XML's ability to encode information in ISO 10646 character sets, providing internationalization support. As a rule of thumb, parameters are encoded in XML entity bodies when they have unbounded length, or when they may be shown to a human user and hence require encoding in an ISO 10646 character set. Otherwise, parameters are encoded within HTTP headers.

In addition to encoding method parameters, XML is used in WebDAV to encode the responses from methods, providing the extensibility and internationalization advantages of XML for method output, as well as input.

As mentioned, WebDAV was designed to provide more methods for handling resources on a server. These additional methods generally require a great deal of information to be associated with both requests and responses to explicitly define the intention of the client or server. The method of communicating all of this information in HTTP was solely the responsibility of the headers in requests and responses. This imposes some limitations on transfers. It is difficult to apply header information to multiple resources in a request and to represent hierarchy.

Because of its inherent extensibility, XML was chosen to describe how these instructions are communicated. XML is crucial to the operation of WebDAV because it provides:

- A method of formatting instructions describing how data is to be handled;
- A method of formatting complex responses from the server;
- A method of communicating customized information about the collections and resources handled; and
- A flexible vehicle for the data itself.

At a high level, a WebDAV instruction processor is really a set of logic that interprets WebDAV methods, followed by an XML parser that interprets the majority of the information communicated.

To sum up, the use of XML in WebDAV turn this technology into such a powerful tool because of the following:

First, XML provides a way of separating data from either the methods that act on that data, or the way the data is presented. This enables straightforward and consistent abstraction of data. For this abstracted data, WebDAV provides a method of consistent, unified transfer between all tiers in the network architecture over channels that are familiar to existing network architectures. This technology enables a much higher level of interoperability between both Microsoft products and third-party applications.

Second, XML enhances WebDAV by providing a means for extensibility. XML allows clients to describe and set properties on a WebDAV server. These properties can then be used to index, search, and process resources on the server. Because of the inherent extensibility of XML, the types of properties and uses for these properties are infinite.

7.4 Differences Of WebDAV From HTTP

Although HTTP 1.1 has support for detecting lost updates through unique identifiers associated with the document state, no support is provided for preventing lost updates in the first place. To solve this problem, WebDAV uses long-duration, whole resource locking as its concurrency control mechanism. The WebDAV protocol provides a write lock, but no read lock capability. On the Web, by default a resource is readable, although it may be protected by access control. Therefore, HTTP doesn't require that a Web browser obtain a lock in order to read a resource, as is the case with traditional database locking, retrofitting the Web with this capability was neither feasible nor desirable. Web servers implement the write operation PUT by saving the contents of the resource in a temporary buffer until the entire new resource has been transmitted, then using internal concurrency control to block read access while the new value is quickly updated. So the traditional database problem of reading a value in an inconsistent state is avoided.

Another traditional database issue, deadlock, is also avoided with WebDAV locks. Since locks are granted via a protocol request, with a given request either granted or denied, there's no blocking, and hence no possibility of deadlock.

WebDAV servers have used differing strategies to implement the features in the protocol. The major difference is the underlying repository chosen by the server to store properties and resources.

7.5 WebDAV Basic Characteristics

WebDAV provides a network protocol for creating interoperable, collaborative applications. Major features of the protocol include (these features are already complete, and expected to be stable):

- **Locking** (concurrency control): long-duration exclusive and shared write locks prevent the overwrite problem, where two or more collaborators write to the same resource without first merging changes. To achieve robust Internet-scale collaboration, where network connections may be disconnected arbitrarily, and for scalability, since each open connection consumes server resources, the duration of DAV locks is independent of any individual network connection.

- **Properties**: XML properties provide storage for arbitrary metadata, such as a list of authors on Web resources. These properties can be efficiently set, deleted, and retrieved using the DAV protocol. DASL, the DAV Searching and Locating protocol, provides searches based on property values to locate Web resources.

- **Namespace manipulation**: Since resources may need to be copied or moved as a Web site evolves, DAV supports copy and move operations. Collections, similar to file system directories, may be created and listed.

WebDAV , under development, extensions:

Several extensions to the base DAV protocol are currently under development in the IETF:

- **Advanced Collections**: this adds support for ordered collections, where the server maintains a single persistent ordering of the URLs in a collection. It also supports referential resources, resources which act like symbolic links in file systems, allowing a client to remotely create a redirect to another resource. This work is expected to be completed in mid-2000.

- **Versioning and Configuration Management**: versioning support, similar to that provided by RCS or SCCS, is the entry level of functionality. The versioning level will support operations such as check-out, check-in, and retrieval of the history list. The ability to directly retrieve a previous version of a resource (allowing links directly to previous revisions) will also be supported. Built on top of the versioning layer is the configuration management layer, which provides support for workspaces and configurations, allowing versioned collections of versioned resources to be worked on. Both layers support parallel development. This work is taking place in the IETF DELTA-V working group, and is expected to be complete in late 2000.

- **Access Control**: the ability to set and clear access control lists. This functionality is crucial for allowing collaborators to remotely add and remove people from the list of collaborators on a single resource. At its most general this activity becomes access control not just for DAV, but for the entire Web.

- **DAV Searching and Locating**

The DAV Searching and Locating (DASL) extension is an application of HTTP/1.1 forming a lightweight search protocol to transport queries and result sets and allows clients to make use of server-side search facilities. DASL defines a new method, SEARCH, to perform the search. The value of properties is increased as DASL provides an efficient an convenient method for searching resources based on their properties.

SOME MORE FEATURES THAT APPEAR

Some other, not so obvious, but still very interesting features of WebDAV are the following:

- WebDAV provides authoring support for Web resources of *any* media type, be they HTML, GIF, JPEG, or other. The versioning extensions also have a primary goal of supporting versioning of *any* media type, not just text-based objects. One consequence of this is that diff and merge operations are not currently expected to be part of the protocol. These operations are the responsibility of the client application, which has deep understanding of the media types it uses.
- WebDAV provides document management capabilities.
- A major feature of DAV is that it provides an upward migration path for existing non-collaborative applications which operate on entire files. WebDAV has been designed so that existing applications can easily add WebDAV support, since locks apply to entire resources, and the namespace operations support "File... Open" and "File... Save" user interfaces. DAV client APIs are lightweight, and hence do not add a significant development burden. The applications in Office 2000 are the first non-collaborative applications to be DAV-enabled, and others will surely follow.
- As applications transition to adding DAV support, existing applications can be used on the local filesystem, and a program like *sitecopy* can upload changed files to a DAV server for the user. Microsoft is also providing a feature called "Web folders" which makes a collection on a DAV server appear to be a directory in Windows. Non-DAV applications cannot work with these folders because the files/folders are only an appearance, and are not mapped into a local filesystem. A Windows "redirector" could be written to make them appear as a local filesystem, but until that time, applications will need to be changed to work with a DAV-enabled server.
- Since DAV works over HTTP, all the benefits of HTTP that FTP cannot provide are provided. For example: strong authentication, encryption, proxy support, and caching. It is true that a user can get some of this through SSH, but the HTTP infrastructure is much more widely deployed than SSH. Further, SSH does not have the wide complement of tools, development libraries, and applications that HTTP does. DAV transfers are also more efficient than FTP. It is possible to pipeline multiple transfers through a single TCP connection, whereas FTP requires a new connection for each file transferred (plus the control connection).
- WebDAV features are designed to accommodate existing tools, making it straightforward to integrate WebDAV-based remote authoring into them. WebDAV's namespace operations provide the ability to create and list collections, and to copy and move Web resources, thus supporting the needs of "File... Open" and "File... Save" user-interface dialog boxes. Locking of entire Web resources provides overwrite protection for all types of Web resources (HTML pages, GIF images, word processing documents, and source-code text files), and in fact, one of WebDAV's design principles is to provide equal support for all Web-resource types.
- Finally, WebDAV also provides support for storing and retrieving metadata, in the form of attribute-value pairs called properties, associated with a resource. The name of a WebDAV property is a URL,

used in this case as a property identifier, not as a locator, and a property value is well-formed XML, gaining XML's advantages for representing structured data and for internationalising string values.

7.6 The Format Of WebDAV Requests

HTTP 1.1 provides a set of methods that clients can use to communicate with servers and specifies the format of responses from servers back to the clients that have issued requests. WebDAV fully adopts all of the methods of this specification, extends some of these methods, and introduces additional methods to provide the functionality described.

The new methods included into the new protocol are the following:

Method	Description
LOCK	This method locks a resource. Using this method does guarantee that no-one except the holder of the lock will be able to modify the resource, as the specification does not require a WebDAV compliant server to use mandatory locks. If used on collections, a *Depth* header may be used to recursively lock all the members in the collection.
UNLOCK	Unlock a resource. Any WebDAV compliant server which implements the `LOCK` is required to support `UNLOCK` too. All general notes for `LOCK` apply to `UNLOCK` as well.
PROPFIND	Retrieve the properties of the resource as identified in the request. A recursive listing is also possible if the resource is a collection itself.
PROPPATCH	Modifies the properties of the resource as identified in the request. As the name suggests, this method does not replace all the properties, but it "patches" the properties by setting/removing individual property elements.
COPY	This method duplicates the source resource identified in the request to the destination as specified by the *Destination* header. No guarantee is made that the resource or properties can be copied identically; the server will make a "best effort". If `COPY` is applied to collections, a *Depth* header with value "infinite " is assumed to be present and so the copy operation will be recursive.
MOVE	This method consist of atomically processed operation a logical equivalent of `COPY` followed by a consistency check and the removal of the source. All general notes for `COPY` apply to `MOVE` as well.
MKCOL	`MKCOL` creates a collection. No recursive operation is supported by this method, that is, the parent collection of the new collection to be created must already exist.
Refinements for methods defined by HTTP/1.1	
PUT	As defined in HTTP/1.1 the `PUT` method will replace the resource if it already exists. The method is not defined for the creation of collections; `MKCOL` is recommended to be used instead.
GET and HEAD	The semantics of `GET` are unchanged when applied to a collection, since `GET` is defined as, "retrieve whatever information (in the form of an entity) is identified by the Request-URI". For this reason, the actual operation that happens in this case is undefined. Likewise `HEAD`

	in not affected since it is defined as `GET` without a message body.
POST	As the operation performed by `POST` depends on the server, no meaningful behaviour can be defined for this method either when applied to collections.
DELETE	If `DELETE` is applied for collections, a *Depth* header with value "infinite " is assumed to be present and so the delete operation will be recursive.

7.7 Managing Documents With WebDAV And XML

EXAMPLES

1) Setting an Author Property Using PropPatch

The following WebDAV request would set an **Author** property on the document entitled Webdav-xml.htm in the collection called *WebDavDocs* on the MyServer.com server:

```
PROPPATCH /WebDavDocs/webdav-xml.htm HTTP/1.1
Host: myserver.com
Content-Type: text/xml
Content-Length: 138

<?xml version="1.0">
<d:propertyupdate xmlns:d="DAV:" xmlns:o="urn:schemas-microsoft-com:office:office">
  <d:set>
    <d:prop>
      <o:Author>Sean Purcell</o:Author>
    </d:prop>
  </d:set>
</d:propertyupdate>
```

The first line of this request specifies the method that the client wishes to enact (**PropPatch**) and gives the absolute URL of the file on which to set the property. The three lines that follow the method are headers that specify the server to which the method will be submitted, and tell the server the type and length of the content to expect.

The XML-encoded body is what tells the server exactly which property to set and the value that should be assigned to it. One important observation in this XML document is the use of namespace declarations. The first attribute in the `<d:propertyupdate>` element defines the use of the WebDAV namespace through the document. To all elements with this prefix throughout the document, the WebDAV-compliant server will know to apply behaviours based on the "*DAV*:" schema. In this case, these specific properties define how to set a property on a document.

The second namespace declaration is for the urn:schemas-microsoft-com:office:office namespace. An excellent strategy to use when designing XML properties is to thoroughly examine existing namespaces for existing properties that could be of use. Equally important, however, is to ensure that the existing property is used in the manner to which it was originally intended to prevent property collision. By using an existing Office property in our scenario, it will allow other clients who recognize this Microsoft custom namespace to interpret this property.

In response to this request, the server would send back a response indicating that the property was successfully set.

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
```

```
Date: Wed, 04 Aug 1999 21:52:58 GMT
Content-Type: text/xml
Content-Length: 310

<?xml version="1.0"?>
<a:multistatus xmlns:b="urn:schemas-microsoft-com:office:office" xmlns:a="DAV:">
  <a:response>
    <a:href>http://myserver.com/WebDavDocs/webdav-xml.htm</a:href>
    <a:propstat>
      <a:status>HTTP/1.1 200 OK</a:status>
      <a:prop>
        <b:Author/>
      </a:prop>
    </a:propstat>
  </a:response>
</a:multistatus>
```

2) Retrieving All Documents in the Collection by Author

A **PropPatch** similar to the one issued above is issued to each of the resources in the \WebDavDocs folder on the server, so that every one of the resources in this collection has associated it with the **Author** property. Now to solve one of the problems outlined in the scenario, the following example retrieves information necessary to populate a table outlining who authored each document in the collection, and the name of each document.

The request to retrieve this information will be the following:

```
PROPFIND /WebDavDocs/ HTTP/1.1
Depth: 1,noroot
Host: myserver.com
Content-Type: text/xml
Content-Length: 184

<?xml version="1.0"?>
<d:propfind xmlns:d="DAV:" xmlns:o="urn:schemas-microsoft-com:office:office">
      <d:prop>
        <d:displayname/>
        <o:Author/>
      </d:prop>
</d:propfind>
```

An additional header that has been added in this request is the depth header, which specifies to which resources the method should be applied. In this case the value "1,NOROOT" specifies that the method should be applied to all immediate children of the specified URL, but not to the URL itself.

3) Searching for Documents by Author

The final problem posed in the scenario is that of searching through a large pool of documents based on the **Author** property. The IETF DAV Searching and Locating (DASL) group is one of the groups that has been formed to extend the functionality provided by WebDAV. This group is concerned with defining a syntax that can be used for searching through WebDAV resources. Because the work of this group has not been finalized, the Exchange team has implemented a **Search** method as part of the WebDAV server component of

Exchange 2000 that uses SQL syntax to perform searches. The following example demonstrates a WebDAV search request for all documents authored by "Sean Purcell" in our collection.

```
SEARCH /WebDavDocs/ HTTP/1.1
Host: myserver.com
Content-Type: text/xml
Content-Length: 295

<?xml version="1.0"?>
<g:searchrequest xmlns:g="DAV:">
  <g:sql>SELECT "DAV:displayname" as prop1,
    "urn:schemas-microsoft-com:office:office#Author" as prop2
    FROM SCOPE('SHALLOW TRAVERSAL OF ".'")
    WHERE "prop2" = 'Sean Purcell'
  </g:sql>
</g:searchrequest>
```

The response to this request returns all of the documents authored by Sean Purcell in the collection, and for each of them returns the **displayname** and **Author** properties tagged by `<prop1>` and `<prop2>`, respectively.

```
HTTP/1.1 207 Multi-Status
Server: Microsoft-IIS/5.0
Date: Wed, 04 Aug 1999 23:56:47 GMT
Content-Type: text/xml

<?xml version="1.0"?>
<a:multistatus xmlns:b="urn:uuid:c2f41010-65b3-11d1-a29f-00aa00c14882/" xmlns:c="xml:"
xmlns:a="DAV:">
  <a:response>
    <a:href>http://myserver.com/WebDavDocs/webdav-xml.htm</a:href>
    <a:propstat>
      <a:status>HTTP/1.1 200 OK</a:status>
      <a:prop>
        <prop1>webdav-xml.htm</prop1>
        <prop2>Sean Purcell</prop2>
      </a:prop>
    </a:propstat>
  </a:response>
  <a:response>
    <a:href>http://myserver.com/WebDavDocs/webdav-http-requests.htm</a:href>
    <a:propstat>
      <a:status>HTTP/1.1 200 OK</a:status>
      <a:prop>
        <prop1>webdav-http-requests.htm</prop1>
        <prop2>Sean Purcell</prop2>
      </a:prop>
    </a:propstat>
  </a:response>
</a:multistatus>
```

7.8 Evaluation Of WebDAV

The biggest advantages of WebDAV come from the fact that the protocol is based on HTTP that already has widely deployed infrastructure. It means that any application built on top of WebDAV can use cryptographically strong authentication, proxying, caching and encryption with SSL, if so desired. The protocol has been developed in co-operation with the major vendors of web publishing tools, which promises a good level of compatibility with existing products.

Another advantage is that WebDAV uses XML, which means that the protocol itself can be extended without fragmenting it (like it happened to HTML). Part of their success of the Web is because the underlying protocols, HTML and HTTP, are simple and yet powerful - and they are based on text, which made them easier to adopt.

On the other hand, the number of drafts and specifications for WebDAV suggests that it is not a simple protocol. Instead of using one method for one function, WebDAV essentially re-invented the HTTP request/response format in XML so it could glue a bunch of requests together. Because of this the protocol is complex and bloated, and middleware and application-level support is also required.

7.9 Participants For WebDAV

While the IETF expects that all participants are individual contributors, a lot can be learned by looking at the corporate affiliations of key participants. After all, if they're able to spend significant time developing DAV, they probably have the full support of their organization.

Participants from Microsoft, Netscape, Novell, and Xerox spent significant time developing the base DAV protocol, and there was significant interest by participants from document management vendors such as Filenet, Documentum and PC Docs as well. Jim Whitehead from U.C. Irvine chaired the effort, which is significant when you realize his officemate is Roy Fielding, a founder of the Apache Group. Combined, these organizations have the ability to make DAV a de-facto standard, with combined controlling market share on both the client and server side.

A team with world-class versioning and CM expertise is working to develop the versioning standard. Participants on the design team come from Rational (ClearCase), Microsoft (Visual SourceSafe), Intersolv (PVCS), Novell (GroupWise), and IBM (VisualWorks), and there are academic participants from U.C. Irvine and Boston University. These people have the experience to develop a good standard, and come from organizations with sufficient market share to make this a de-facto standard. These organizations also have significant experience and market share in software development tools. However, it is important to know that the IETF process does not have a notion of membership, and there are no corporations -- only individual contributors (who sometimes happen to have the institutional backing of their corporation, a nice coincidence). There is no voting, only rough consensus on the mailing list. The result of this is that individual contributors (with or without corporate affiliation) can and do make a big difference, and are one of the primary reasons why IETF standards typically are of high technical quality.

7.10 Software

Clients

There a number of web authoring tools that support distributed authoring, either by proprietary server extensions or by using the PUT method described in HTTP/1.1. These include Microsoft Frontpage, AOLPress and Netscape Communicator. Few products are WebDAV-compliant yet, clients that support WebDAV now or in the future include Microsoft Internet Explorer 5, NetObjects Fusion and Microsoft Office 2000. A number of Open Source solutions also exist.

Web Servers

- Zope is an Open Source Web server which supports WebDAV Class 1 as defined in the specification.
- The coming version Microsoft Internet Information Server 5 will also support WebDAV.
- Apache, the most popular Web server, does not fully support HTTP/1.1 and contains no support for WebDAV. Two modules exist that can add distributed authoring support to Apache: *mod_put* and *mod_dav*. The former only implements PUT and DELETE, so that HTTP/1.1-aware Web clients can be used for authoring, while the latter is aimed to provide full WebDAV compliance for Apache.

7.11 Current Projects

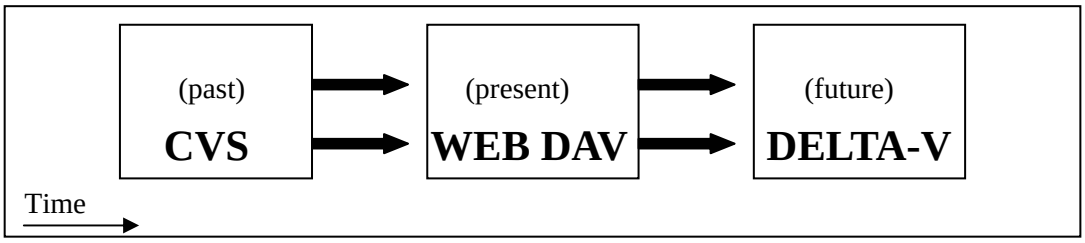
There are quite a few projects that are being developed. Here are four that are active:

- **mod_dav**: a DAV module for Apache, which was mentioned just above;
- **sitecopy**: a tool for copying web sites from a local system up to a web server;.
- **cadaver**: similar to a command-line FTP tool, this tool provides for easy, direct interaction with a DAV server; and
- **Goliath**: a DAV client application and a DAV library for the MacOS operating system.

7.12 The Future Of The WebDAV

The future of WebDAV is definitely called *Delta-V*. Delta-V is an activity which extends WebDAV (and HTTP) by adding versioning support to WebDAV, this is the new feature that covers the need of preserving the history of work.

So, the history of distributed software development consists of three basic parts:



Delta-V is under development at present time and the work is driven by a working group of participants who come from the leading companies of document management and web portal system areas: IBM, Microsoft, Datachannel, Novell, Rational, Merant, Object Technology International and Dynamic Diagrams, with university participation from U.C Irvine.

Delta-V is web oriented. The primary advantage seems to be exactly this kind of tight integration with the web. With Delta-V, Web resources are edited in-place, at a specific URL, and no mapping of filenames to URLs is necessary. Furthermore, the Web-native Delta-V protocol can handle the different types of Web resources better than a file-oriented system like CVS. By versioning Web resources, Delta-V allows HTML links to old revisions of Web pages, creating a sort of time machine for the Web. Linking to a specific revision often can preserve the semantic meaning of a link, such as when linking to a Web-log site that changes frequently, where the linked-to information may be gone in a week. If the site used Delta-V to version its content, these old revisions would still be accessible.

The Delta-V protocol has several unique features. Delta-V assumes that most editing will take place directly on Web resources, which differs from CVS in that there's no local replica. Isolation from the changes of other team members is provided by "workspaces," which provide each collaborator with his or her own view on the resources being edited. Unlike the local replicas that provide isolation in CVS, workspaces isolate collaborators as they work on the remote Web server. Overwrite conflicts are avoided because a resource can be checked out by multiple people simultaneously, and each check out creates a separate working resource. Each collaborator actively working on a resource has a separate virtual working area, identified by his or her workspace, and modifications are made first in a workspace, then merged with the changes of other collaborators.

Delta-V provides a cross-platform integration layer, thus bringing the benefits of remote Web collaboration support to a diverse set of existing back-end repositories that do not currently provide Web authoring or versioning support. Judging by the participants in the working group, the Delta-V protocol will be mapped to SCM systems, document management systems, and content management systems, all of which employ a database to provide their features. This makes the Delta-V protocol a more powerful data integration technology than the CVS client/server protocol, which maps only to the CVS repository.

Delta-V provides versioning of collections, a feature not supported by CVS. When a collection is versioned, collections and their contents follow the check-out/edit/check-in model. When a collection is checked in, its membership is frozen, and can't be changed until the collection is checked out again. Making a new file or deleting an existing file requires the parent collection to be checked out. When all collections in a project are versioned, it's possible to record permanently the membership of each collection for each moment in time, thus making configuration management support possible. Once both collections and their contents are versioned, it's possible to explicitly pick a single revision of each collection and file (often the most recent revision), creating a snapshot of the entire project.

CVS doesn't provide full versioned collection support, leading to odd glitches. As an example, consider renaming a file from A to B. Using CVS, this requires three steps: copying file A's contents into the new location at B; using a <code>cvs add</code> to put B into the CVS repository; and a <code>cvs remove</code> to delete file A. If the collection containing B were reverted to a previous state when A was present but B had not yet been added, the collection will contain both A and B. Since CVS doesn't store previous revisions of collections, it doesn't know when B was added, and so can't revert the collection correctly. Because Delta-V versions collections, it can avoid this problem. Renaming the file in Delta-V would involve checking out the collection to make it editable, moving the file from A to B, and then checking in the collection. If the collection is reverted to the original version, just before the initial check out, it will contain A, but not B, and similarly the following revisions will contain B, but not A. Versioned collections thus provide the foundation for rigorous configuration management.

Since Delta-V assumes work will take place directly on a Web server, rather than on a local replica, existing WebDAV editing tools, like Office 2000, that are not versioning-aware need to be accommodated. Delta-V can automatically record, as separate revisions, changes to a document made by a versioning-unaware client. Delta-V also divides its functionality into two layers: a simple versioning layer, and a more complex SCM layer. Since authoring clients (word processors, text editors, spreadsheets, and so on) typically work on a single file at a time, they are only expected to use the basic versioning layer to support a check out/edit/check in style of work. The typical authoring client is not expected to provide a user interface for operations like creating and reverting configurations, since a configuration spans an entire project, far greater than their single-file editing scope. A separate SCM control panel application will make use of the features in the SCM layer. This control panel will operate at a collection and project level, providing the capability to create a project configuration or revert to a previous configuration. It will complement the single-file focus of the authoring tools with project-wide capabilities. A full-featured programming environment will be a third class of Delta-V application, one that uses both the versioning and configuration capabilities of Delta-V, providing support for editing individual source-code files, as well as project-level SCM support.

Despite their differences, Delta-V and CVS have much to offer each other. Though Delta-V has been designed for collaborators to work directly on a Web server, it's technically feasible to use the protocol to create local replicas, as in CVS. In fact, though it has not been attempted, it appears to possible to replace the CVS client/server protocol with Delta-V, and an existing WebDAV client called sitecopy provides a glimpse of how this could be done. The sitecopy utility allows a local file-system directory to be replicated to a remote WebDAV server, so a Web site can be created locally using file-system based authoring tools, then published remotely using the WebDAV protocol. In its remote replication support, sitecopy is similar to the CVS update operation. Though sitecopy and WebDAV don't support versioning, it's not a far stretch to imagine adding bi-directional synchronization, conflict flagging, and versioning operations to sitecopy, thus creating a system that has many of the capabilities of CVS. But why recreate the CVS user interface? It's far better to integrate the Delta-V protocol into CVS, retaining the benefits of the CVS without having to learn a

new system. Since Delta-V can map to multiple back-end repositories, Delta-V would allow the CVS style of work to be used against multiple repositories, not just with CVS.

The Delta-V protocol opens up several intriguing possibilities for building software systems. These possibilities vary based on where the source code, compiler, and object files are located -- on the remote Delta-V server or on the local machine. If they're all on the local machine, then the build process is very CVS-like, with source code replicated to the local machine before the compiler begins operation, yielding object files that reside locally. But if the source code, compiler, and object files are held remotely, a client would initiate a build by sending a build request to a remote compile server, giving the URL of a makefile and a workspace, storing the object files in the same version-controlled URL hierarchy as the source code. In this scheme, a different compile server could compile each platform variant. While the compiler wouldn't typically be placed on the same machine as the Delta-V server -- so compiles don't adversely affect server performance -- it would be reasonable to place the compile server on the same local storage area network as the Delta-V server. Many interesting configurations are possible for build management using Delta-V, undoubtedly an area where implementations will innovate on different strategies.

With a proven track record based on successful use on a wide range of Open Source projects, CVS is a low-cost, high-value system available today. Looking to the future, the Delta-V protocol melds versioning and SCM with the Web, adding powerful team collaborative work facilities, with the potential for a value-adding integration with CVS.

Whether you're looking at the state of things today, or the promise of the future, the implication of these two technologies is clear:

‘’It's easier than ever before to assemble a virtual team for remote collaborative project work’’.

8. RSS

8.1 Introduction in RSS

Sharing content among sites is most often called syndication. Providing content from one source for distribution in many different channels is what a syndicate does, and it usually requires an established business relationship. Web offers a new open-ended syndication model that's hardly traditional.

The basis for this new model is the XML-based format known as **Rich Site Summary** (RSS). The RSS format was originally developed by Netscape for use on My Netscape Network, a customisable start page for Netcenter. My Netscape Network provides a simple RSS framework for web sites to create channels that can then be added to the customisable start page.

Today more and more content providers have been adopting RSS as a means of circulating headlines and links to new stories on their sites. RSS is becoming a vital "What's New" mechanism that serves a variety of purposes while helping to attract traffic from many different locations on the Web. RSS is a better way to share data than more common approaches, such as fetching and parsing HTML, or using proprietary APIs, database dumps, and cobranding (a method in which the information provider hosts custom versions of the application for each customer).

Under the RSS model, each site publishes a file describing the contents of its "channel." Other sites can subscribe to that channel and grab its contents. The RSS file could be converted to HTML and displayed directly on a subscriber site, or it might be edited first to select only those items that are appropriate for the site's audience. [WIGG99],[EISE00]

8.2 RSS Syntax – Elements

RSS is a simple mark-up language based on XML. Specifically, RSS is an XML vocabulary for describing metadata about websites. That means that an RSS file contains placeholders for data, which are identified by a starting and ending tag. An RSS file is usually generated by a simple program but it can also be created by hand. An RSS file must use the RSS tags properly and also have the correct structure.

Like any XML document, the first line of an RSS file contains an XML declaration that isn't required, but is recommended for backwards compatibility. The next item in an RSS file is the DTD that identifies the file as an RSS document. This is necessary to determine whether the file is valid when tested against the rules of the RSS DTD.

The `rss` element is the root or top-level element of an RSS file. The `rss` element must specify the `version` attribute. (The current version is `0.91`). It may also contain an `encoding` attribute (the default is `UTF-8`). In an RSS file a language code must be specified, which reflects the language in which the channel content is written. Only one language code can be specified per channel. The specified language code enables Netscape to classify the channel with other channels in the same language and it enables Netscape to encode the channel properly in the browser window.

Within the `rss` root element of an RSS file, there is one and only one `channel` element, which contains all the other elements. There are four main sections of an RSS file:

- information about the channel;
- information about an optional channel image;
- up to 15 channel items; and

- an optional form input box.

Each `channel` element must contain the following elements:

- `title` - the name of the channel;
- `description` - a short description of the channel;
- `link` - an HTML link to the channel Web site;
- `language` - the language encoding of the channel; and
- one or more `item` elements.

A `channel` may also contain the following optional elements:

- `rating` - the PICS (Platform for Internet Content Selection) rating for the channel Web site. PICS ratings are assigned by an independent agency;
- `copyright` - content copyright;
- `pubDate` - date the channel was published;
- `lastBuildDate` - date the RSS was last updated;
- `docs` - additional information about the channel;
- `managingEditor` - channel's managing editor;
- `webMaster` - channel Webmaster;
- `image` - channel image;
- `textinput` - allows a user to send an HTML form text input string to a URL;
- `skipHours` - the hours that an aggregator should not collect the RSS file; and
- `skipDays` - the weekdays that an aggregator should not collect the RSS file.

A channel may contain an image or logo. The `image` element must contain the image title, commonly used as the ALT attribute when converted to an HTML image element, and the URL of the image itself. The image element may also include the following optional elements:

- `link` - a URL that the image should be linked to;
- `width` - the image width;
- `height` - the image height; and
- `description` - an area for additional text.

An `item` has three elements — a title, a link, and a description. The only requirements for these elements are that the contents of the `title` element must be less than 100 characters long, and the contents of the `link` and `description` elements must be less than 500 characters long. The `description` element is optional. There are a maximum of 15 items allowed per channel. The `item` elements provide the headlines and summaries of the content one wants to share with other sites.

The form input box can be used to solicit user feedback, provide a search interface, or perform any HTML form action that uses a single text field and a single submit button. The `textinput` element lets users input data in an HTML text field:

- `title` - label of the submit button;
- `description` - text input description;
- `name` - text input name; and
- `link` - URL to which to send the input. [WIGG99],[EISE00]

For a complete table with the RSS 0.91 tags and syntax as well as examples of usage see at Netscape tutorial: http://my.netscape.com/publish/help/mnn20/quickstart.html. The RSS version 0.91 DTD resides at: http://my.netscape.com/publish/formats/rss-0.91.dtd.

8.3 Well-formed RSS files

RSS 0.91 supports all legal HTML and decimal Entities. Decimal entities have the format `&#ref;` where `ref` is a number that references the character (`#ref` must be between 0-255). Entity references have the format `&nnn;` where `nnn` is a text string that references the character. XML also has reserved characters, such as angle brackets, for which a decimal or entity reference must be used. [NETS99]

The RSS specification includes all HTML entities for convenience; however, no HTML elements can be included. The RSS file is valid if only the elements defined in the specification have been used. An XML parser can't properly parse an XML file unless it follows the following well-formed rules:

- Each starting tag must have an ending tag;
- Internal entities such as `&`, `&quot;`, `<`, `>`, must be encoded; and
- XML elements must be well balanced; that is, the end tag should be at the same level in the tree as the start tag. [EISE00]

8.4 Uses of RSS

For some time, the primary use of RSS (Rich Site Summary) has been to allow My Netscape users to select a channel to display on their personalized page. When users register with My Netscape, they are given the option of configuring their start-up page with several common types of news and information channels. In addition to personalizing Netscape pages, there are other, possibly more powerful, uses for RSS.

8.4.1 Syndication

Once an RSS file exists, any other site can grab it regularly. RSS standardizes a format for the delivery of content. This makes it easier for a content provider to distribute content broadly, and for an affiliate to receive and process content from multiple sources. However, in most cases, the actual content is not really distributed, only the headlines are, which means that users will come back to the affiliate site if they're interested in the story. For example, many content providers use ad banners as a primary source of revenue. This model depends on a large volume of users reading their content on a regular basis. The RSS format is a marriage made in heaven for extending readership. This explains why most early adopters have been news providers.

Once an RSS file is being published, content can begin to flow into new venues such mailing lists, PDAs, cell phones, and set-top boxes. For example, anyone may decide to offer headlines in a PDA-friendly format, or

create a weekly email newsletter comprising what's new on their Web site. He can also flow data between partners or affiliate Web sites.

8.4.2 Aggregation

The practice of gathering multiple RSS channels into one central location is called aggregation. While most aggregator Web sites share a common goal (gathering content) they serve different purposes. For example, *my.netscape.com* offers its feeds as channels to Netcenter users, whereas *iSyndicate.com* offers news feeds primarily for use on other Web sites and *my.userland.com* offers a service similar to my.netscape.com. However, the aggregator also offers aggregate feeds, which send new content to partners via XML-RPC function calls. The benefit of using aggregators is that they make many feeds available from one place. Furthermore, an aggregator may offer tools or solutions that allow partners to customize feeds and minimize the integration effort. In addition, an aggregator site might provide tools and services that make it easier for content providers to syndicate their information. Another key reason for using RSS is the ability to share data between web sites. Just as anyone can share data between their site and an aggregation site, so can they share data on a one-to-one, or one-to-many, basis between groups of partner web sites.

8.4.3 Weblogs

A Weblog is a portal to the life of an individual or group. The ideas posted on a Weblog often include personal, political, technical, or editorial comments that are significant to the author. It turns out that RSS is a good foundation for creating a Weblog. An example is *PerlXML.com*, a site containing Perl/XML resources and news. A simple CGI script that uses the XML::RSS Perl module is used to add new headlines. The script updates both the front-page HTML file and the RSS headlines, which are then picked up by several aggregators including my.netscape.com and my.userland.com. This dual-purpose method alleviates the Weblog editor from updating multiple files. Instead, editors can focus on their job and let an application on the Web server do the work behind the scenes. [EISE00], [WIGG99]

8.4.4 RSS Channel Servers

The following sites provide access to RSS Channels. They may or may not accept custom RSS Channels designed by third parties. [MART00]

- My Christian Start (http://mychristianstart.com/)

  A Christian-oriented start page server that still offers access to many channels through the XMLTree.

- My Netscape (http://my.netscape.com/)

  Affiliated with Netscape Center, this service allows members to create customisable start pages, which are loaded with channels provided by Netscape and third-party providers.

- My Userland (http://my.userland.com/)

  Less sophisticated than MyNetscape, this service allows members to create customisable start pages loaded with channels. All channels appear to be submitted by third parties.

- My Web News (http://www.mywebnews.com/default.asp)

  This service allows members to create customisable start pages. They call the RSS Channels "modules".

- Starts Here.Net (http://www.startshere.net/)

  Headline service that uses RSS channels to serve up their content. Third party channels may be submitted.

- The XML Tree (http://www.xmltree.com/)

This is a directory of news scripting and RSS documents used to create channels for Web page servers and email newsletters. The directory captures content and creates "leaf nodes" in the tree. It also provides users with the option of adding RSS Channels that they find to start pages on various XML servers.

- Moreover.com (http://www.moreover.com/)

Another aggregation site where someone can register his RSS file.

8.5 RSS Tools

- An RSS File Validator Tool, a tool that scans an RSS file and alerts the user to any errors it finds, is provided from Netscape and can be accessed at this URL:

  http://my.netscape.com/publish/help/validate.tmpl

- Site Summary allows to construct an RSS file, suitable for inclusion as a My Netscape channel. The Site Summary object allows presenting site summary as an XML file, for use with My Netscape or My Userland or as an HTML preview. Additionally, Site Summary can produce the file automatically from objects in the same folder as it. Site Summary also supports import of external RSS files, so users can get headlines from any site that uses RSS integrated into their web site.

  http://www.zope.org/Members/edmundd/SiteSummary/

- The RSS Channel Editor is a simple Perl CGI script that makes it easy to maintain an RSS channel. It will retrieve an RSS channel given a URL and save it locally, via the Web browser, or remotely, on the Web server it's running on. By Jonathan Eisenzopf.

  http://www.webreference.com/perl/tools/rssedit.zip

- RSS Maker is a Web based tool that makes it easy to create and maintain RSS files. By Jonathan Eisenzopf.

  http://www.webreference.com/perl/tools/makerss.pl

- XML::RSS 0.8. This Perl module provides a basic framework for creating and maintaining Rich Site Summary (RSS) files. By Jonathan Eisenzopf

  http://www.perlxml.com/modules/XML-RSS-0.8.tar.gz

  Tutorial: http://www.webreference.com/perl/tutorial/8/

- Perl script for converting an RSS file to an HTML file. It takes one argument, either a file on the local system, or an HTTP URL. By Jonathan Eisenzopf.

  http://www.webreference.com/perl/rss2html.pl

- Carmen's Headline Viewer is a Windows application that displays RSS content without a browser.

  http://www.vertexdev.com/HeadlineViewer/

8.6 Future of RSS

RSS can be used easily as a generic format for exchanging content on the Web. More Web sites are using XML and RSS as they discover that the technologies help promote traffic to a site. RSS is a good starting point towards complete adoption of XML. However, while RSS is capable of syndicating content headlines, there are other XML formats like XMLNews and ICE (Internet Content Exchange) that are better suited for handling larger syndication systems. [EISE00]

Specifically about Netscape, in the last release of My Netscape Network (MNN), it has limited the complexity of the RSS format. However, in the future, as proclaimed in the company's site, Netscape plans to work with other organizations to enhance the RSS format to support additional tags, while making it more compliant with existing W3C standards for XML and RDF (Resource Description Format).

Netscape is planning to enhance RSS to support a variety of new features. Such features include a richer metadata description, for more powerful searching and personalization capabilities, sub channels, related channel information, keyword-based collection of data, direct referencing of other RDF structures, PICS ratings, and more. [NETS99]

9. RDF

9.1 Introduction in RDF

RDF (Resource Description Framework) is a framework for describing and interchanging **metadata.** It provides interoperability between applications that exchange machine-understandable information on the Web. RDF emphasizes facilities to enable automated processing of Web resources. Metadata is "data about data" or specifically in this context "data describing web resources." The distinction between "data" and "metadata" is not an absolute one; it is a distinction created primarily by a particular application ("one application's metadata is another application's data").

RDF is a collaborative design effort; no one individual or organization invented RDF. Several W3C Member companies contribute intellectual resources to its development. The development of RDF as a general metadata framework - and in a way as a general *knowledge representation mechanism* for the web - was heavily inspired by PICS. While RDF started as an extension of the PICS content description technology, it also draws upon the XML design as well as technology submissions by Microsoft (XML Web Collections) and Netscape (XML/MCF). Other documents, such as Microsoft's XML-Data paper, Site Map proposals, and the Dublin Core/Warwick Framework have also influenced the RDF design. . [SWIC99] [LASS97]

RDF encourages the view of "metadata being data" by using XML as its encoding syntax. The resources being described by RDF are, in general, anything that can be named via a URI. The broad goal of RDF is to define a mechanism for describing resources that makes no assumptions about a particular application domain, nor defines the semantics of any application domain. The definition of the mechanism should be domain neutral, yet the mechanism should be suitable for describing information about any domain.

At the core, RDF data consists of *nodes* and attached *attribute/value pairs*. Nodes can be any web resources, even other instances of metadata. Attributes are named properties of the nodes, and their values are either *atomic* (text strings, numbers, etc.) or other resources or metadata instances. In short, this mechanism allows building *labelled directed graphs*. The essence of RDF is the model of nodes, attributes, and their values. In order to store instances of this model into files or to communicate these instances from one agent to another, we need a graph serialization syntax. RDF and XML are complementary. There will be alternate ways to represent the same RDF data model, some more suitable for direct human authoring.

RDF in itself does not contain any predefined **vocabularies** for authoring metadata. Some of the vocabularies in the foreseeable future are a PICS-like rating architecture, a digital library vocabulary (referred as "*Dublin Core*"), and a vocabulary for expressing digital signatures. Anyone can design a new vocabulary; the only requirement for using it is that a designating URI is included in the metadata instances using this vocabulary. This use of URIs to name vocabularies is an important design feature of RDF: many previous metadata standardization efforts in other areas have foundered on the issue of establishing a central attribute registry. RDF permits a central registry but does not require one. [LASS97]

9.2 Features of RDF

RDF provides the following **features**:

- interoperability of metadata;
- machine understandable semantics for metadata;
- better precision in resource discovery than full text search; and

- future-proofing applications as schemas evolve.[SWIC99]

RDF is built on the following **rules**:

1. A **Resource** is anything that can have a URI. This includes the entire world's Web pages, as well as individual elements of an XML document. An example of a resource is a file whose URL is `http://www.aueb.gr/~p3960134/test.rdf`

2. A **PropertyType** is a Resource that has a name and can be used as a property, for example `Author` or `Title`.

3. A **Property** is the combination of a Resource, a PropertyType, and a Value. The Value can just be a string, e.g. "The Author of `http://www.aueb.gr/~p3960134/test.rdf` is John Doe" or it can be another resource, e.g. "The Home-Page of `http://www.aueb.gr/~p3960134/test.rdf` is `http://www.aueb.gr`".

4. There is a straightforward method for expressing these abstract Properties in XML, for example:

```
<RDF:Description href='http://www.aueb.gr/~p3960134/test.rdf'>
<Author>John Doe</Author>
<Home-Page RDF:href='http://www.aueb.gr' />
</RDF:Description>
```

RDF is carefully designed to have the following characteristics:

- **Independence** : Since a PropertyType is a resource, any independent organization (or even person) can invent them.
- **Interchange** : Since RDF Properties can be converted into XML, they are easy to be interchanged.
- **Scalability** : RDF properties are simple three-part records (Resource, PropertyType, Value), so they are easy to handle and look things up by, even in large numbers.
- **PropertyTypes are Resources** : It means that they can have their own properties and can be found and manipulated like any other Resource.
- **Values Can Be Resources** : For example, most Web pages will have a property named Home Page that points at the home page of their site. So the values of properties, which obviously have to include things like title and author's name, also have to include Resources.
- **Properties Can Be Resources** : So they can have properties too. It is necessary often to do lookups based on other people's metadata. One useful way to do this would be with metadata; so Properties will need to have Properties. [BRAY98]

9.3 RDF & XML

XML allows inventing tags, and for the tags to contain both text data and other tags. Also, XML has a built-in distinction between element types and elements. This corresponds naturally to the distinction between PropertyTypes and Properties. So it seems as though XML documents should be a natural vehicle for exchanging general-purpose metadata.

XML, however, falls apart on the **Scalability** design goal. There are two problems:

1. The order in which elements appear in an XML document is significant and often very meaningful. This seems highly unnatural in the metadata world. Furthermore, maintaining the correct order of millions of data items is expensive and difficult, in practice.
2. XML allows complicated constructions, which when are represented in computer memory, result in data structures that mix trees, graphs, and character strings. In general, these are hard to handle in even moderate amounts.

On the other hand, something like XML is an absolutely necessary part of the solution to RDF's **Interchange** design goal. XML is unequalled as an exchange format on the Web. But by itself, it doesn't provide what is necessary in a metadata framework. [BRAY98]

9.4 Uses of RDF

RDF metadata can be **used** in a variety of application areas. For example:

- in *resource discovery* to provide better search engine capabilities;
- in *cataloguing* for describing the content and content relationships available at a particular Web site, page, or digital library;
- by *intelligent software agents* to facilitate knowledge sharing and exchange;
- in *content rating*;
- in describing *collections* of pages that represent a single logical "document";
- for describing *intellectual property rights* of Web pages; and
- RDF with *digital signatures* will be key to building the "Web of Trust" for *electronic commerce*, *collaboration*, and other applications. [SWIC99]

9.5 Storing RDF in a relational database

There is a need for persistent storing and manipulation of large amounts of RDF data. One of the alternatives to do that is to use the relational database technology. A major advantage of this approach is that is provides a scalable off-the-shelf solution. A list of the criteria to be considered for the database schema design could be the following:

- Scalability;
- Querying (queries supported, easily formulated and processed);
- Efficiency (cost of delivering the result);
- Optimisation; and
- Organization (overhead associated with storing the data).

The following is probably the simplest way of dealing with RDF one could think of. Many applications that need a simple RDF store could do with this solution:

```
CREATE TABLE triple (
  property varchar(255),
  resource varchar(255),
  value blob,
  hint char(1)
);
```

The terms property, resource and value have already discussed before, while blob stand for binary large object. This approach takes advance of the fact that RDF properties are simple three-part records (Resource, PropertyType, Value). [MELN00]

9.6 Future

Further development will enable RDF to also provide:

- a uniform query capability for resource discovery;
- a processing rules language for automated decision-making about Web resources; and
- language for retrieving metadata from third parties.

In general, RDF provides the basis for generic tools for authoring, manipulating, and searching machine understandable data on the Web thereby promoting the transformation of the Web into a machine-processable repository of information. RDF has the potential to bring dramatic benefits to the worlds of searching, retrieval and many other aspects of content automation. [SWIC99], [BOYE98]

Once the web has been sufficiently "populated" with rich metadata, searching on the web will become easier as search engines have more information available, and thus searching can be more focused. Doors will also be opened for automated software agents to roam the web, looking for information for users or transacting business on their behalf. The web of today, the vast unstructured mass of information, may in the future be transformed into something more manageable - and thus something far more useful. [LASS97]

APPENDIX A : COMPARISON OF XML WITH OTHER SIMILAR LANGUAGES

A.1 Comparison with SGML

SGML is the Standard Generalized Mark-up Language defined by ISO 8879. It is the international standard for defining descriptions of the structure and content of different types of electronic document. SGML is the "mother-tongue" for both XML and HTML. XML is defined as an application profile of SGML. What this means is that XML documents are conforming SGML documents and that any fully conformant SGML system will be able to read XML documents. However, using and understanding XML documents *does not* require a system that is capable of understanding the full generality of SGML. XML is considered to be a **restricted form** of SGML. XML omits the more complex and less-used parts of SGML in return for the benefits of being easier to write applications for, easier to understand, and more suited to delivery and interoperability over the Web [UCC98]. XML simplifies SGML by capturing about 80 percent of SGML's functionality with only 20 percent of the complexity[MIKO98].

SGML has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade, but it is not well suited to serving documents over the web. SGML provides arbitrary structure, but is too difficult to implement just for a web browser. Full SGML systems solve large, complex problems that justify their expense. Viewing structured documents sent over the web rarely bear such justification. While XML is being designed to deliver structured content over the web, some of the features it lacks make SGML a more satisfactory solution for the creation and long-time storage of complex documents. In many organizations, filtering SGML to XML will be the standard procedure for web delivery.

A.2 Comparison with HTML

Presentation is an important aspect of the **markup languages**. The best way to use markup is by a formal separation of structure and presentation. HyperText Markup Language (HTML) was an SGML DTD with structure (`\begin ... \end`) and semantics (implicitly presentation-oriented with element names like `table`) inspired by the LATEX environment. Initially, HTML did not have any explicit presentational support for defining the appearance of documents, a task that was provided by settings in the Web browser. As the demand from the language increased, ad-hoc and at times proprietary extensions to support presentation, interaction (forms), mathematical notation, metadata, multimedia and navigation (frames) were added to it. A major problem that resulted from this effort was a mixture of structure and presentation. The lack of a clear structure and primarily presentation-oriented semantics made the task of robot indexing and searching imprecise. Furthermore, inspite of the extensions to HTML it could not serve all possible domains of application.[KAM00]

In HTML, both the tag semantics and the tag set are fixed whereas in XML neither semantics nor a tag set is specified. For example an `<h1>` tag is always a first level heading in HTML, while the tag `<productid>` is meaningless. HTML is heavily document-structured and presentation-oriented. HTML has serious drawbacks that make it a poor fit for designing flexible, powerful, evolutionary information systems:

- **HTML isn't extensible:** It doesn't allow application developers to define custom tags for application-specific situations;

- **HTML is very display-centric:** It is useless for other common network applications like data replication or application services;
- **HTML isn't usually directly reusable:** It cannot specify data presentation in terms of structure, so that when data are updated the formatting can be "reapplied" consistently and easily;
- **HTML only provides one 'view' of data:** It cannot display the same data in different ways based on user requests; and
- **HTML has little or no semantic structure:** Most Web applications would benefit from an ability to represent data by meaning rather than by layout. [JOHN00]

XML differs from HTML in three major respects:

- **Extensibility**: HTML does not allow users to specify their own tags or attributes in order to parameterise or otherwise semantically qualify their data while with XML information providers can define new tag and attribute names specific to their applications;
- **Structure**: HTML does not support the specification of deep structures needed to represent database schemas or object-oriented hierarchies while XML document structures can be nested to any level of complexity; and
- **Validation**: HTML does not support the kind of language specification that allows consuming applications to check data for structural validity on importation while any XML document can contain an optional description of its grammar for use by applications, that need to perform structural validation.[BOS97]

The W3C, in conjunction with browser vendors and the WWW community, is constantly working to extend the definition of HTML to allow new tags to keep pace with changing technology and to bring variations in presentation (style sheets) to the Web. However, these changes are always rigidly confined by what the browser vendors have implemented and by the fact that backward compatibility is paramount.

The **advantages** of XML instead of HTML could be generalized as following:

- Authors and providers can design their own document types using XML, instead of being confined to HTML;
- The hypertext linking abilities of XML are much greater than those of HTML;
- XML can provide more and better facilities for browser presentation and performance;
- Information can be more accessible and reusable, because the more flexible mark-up of XML can be used by any XML software instead of being restricted to specific manufacturers as has become the case with HTML; and
- XML files can be used outside the Web as well, in an SGML environment. [UCC99]

XML can encode the representation for ordinary documents, structured records, objects, with corresponding data and methods, meta-content about a Web site, graphical presentations. The flexibility of a single data representation format allows any software to determine the semantics of a data element, without previous knowledge of the underlying meaning of the data. [SUSA97]

Some **benefits** an author might have by representing information in XML:

- XML is at least as readable as HTML and probably more so;
- XML is more versatile than HTML;

- The tags don't have anything to do with how the document is displayed. The markup indicates what the information *means,* since the formatting information for an XML file is usually written in a *style language* and stored separately from the XML;
- A lot of the programming is already done. Much of the error checking for the validity of the input is done by the parser; and
- If a system needs to interoperate with other systems, a standard DTD (like XML/EDI, for example) can be chosen so that other systems will automatically understand the system's vocabulary, and vice versa. [JOHN00]

Since there is a large volume of HTML files over the Internet and there are not yet native XML browsers, it could be useful to somehow insert XML data in a HTML file. A **data island** is an XML document that exists within an HTML page. [MICR] This is a means to embed fragments of XML in HTML files without having to load it through script or through the <OBJECT> tag. Almost anything that can be in a well-formed XML document can be inside a data island. The XML elements marks the beginning of the data island, and its ID attribute provides a name to reference the data island.

The XML for a data island can be either inline:

```
<XML ID="TEST">
  <product>
    <name code=”13342”>printer</name>
  </product>
</XML>
```

or referenced through a SRC attribute on the XML tag:

```
<XML ID="TEST" SRC="product.xml"></XML>
```

The SCRIPT tag can also be used to create a data island:

```
<SCRIPT LANGUAGE="xml" ID="TEST">
    <product>
    <name code=”13342”>printer</name>
  </product>
</SCRIPT>
```

XML itself does not replace HTML: instead, it provides an alternative that allows users to define their own set of markup elements. HTML is expected to remain in common use for some time to come and Document Type Definitions for HTML will be available in XML versions as well as in original SGML. XML is not backwards compatible with existing HTML documents, but documents conforming to HTML3.2 can easily be converted to XML.

So, in order to convert HTML files to XML, someone should make them first well-formed DTD-less documents and then write a style sheet. HTML files converted to XML format currently have to be DTDless because there are few working XML versions of the current SGML-based HTML DTDs yet. It is necessary to convert existing HTML files to be well-formed because XML does not allow end-tag minimization (missing `</p>`, *etc*), which is allowed in most HTML DTDs. [UCC99]

The Extensible HyperText Mark-up Language (**XHTML**) is a W3C project. XHTML 1.0 is a reformulation of HTML 4.0 as an XML 1.0 application. There are three DTDs corresponding to the ones defined by HTML 4.0. The semantics of the elements and their attributes are defined in the W3C Recommendation for HTML 4.0. These semantics provide the foundation for future extensibility of XHTML. Compatibility with existing HTML user agents is possible by following a small set of guidelines. [From the XHTML Specification]

Summarizing, it could be pointed out that XML is an important **development** because it removes two constraints that are holding back Web developments:

1. dependence on a single, inflexible document type (HTML),
2. the complexity of full SGML, whose syntax allows many powerful but hard-to-program options.

XML simplifies the levels of optionality in SGML, and allows the development of user-defined document types on the Web. XML should be considered as being $SGML^{--}$ rather than $HTML^{++}$, in an effort to adopt the meaning of the corresponding unary operators of C language [UCC99].

APPENDIX B : COMPARATIVE OVERVIEW OF XML BROWSERS

The following chart compares the existing browsers, Netscape 6 Preview Release 1, Opera 4 beta 3, Internet Explorer 5.0/Mac, and Internet Explorer 5.5/Win, regarding their abilities in supporting properly the display of XML documents [LAUR00] :

	Browser			
Feature	**Netscape 6 Preview Release 1**	**Opera 4 beta 3**	**Internet Explorer 5.0/Mac**	**Internet Explorer 5.5/Win**
XML display without style sheet	Everything as inline	Everything as inline	Structured presentation	Structured presentation
Uses W3C PI to connect style sheets	yes	yes	yes	yes
`display:inline`	yes	yes	yes	yes
`display:block`	yes	yes	yes	yes
`display:none`	yes	yes	yes	yes
`display:list` (and `list-item`)	yes	yes (numbering issues)	yes (numbering issues)	no
`display:table` (and contents)	yes	yes (some interactions with embedded HTML)	no	no
XLink support	yes (3/3/98 draft)	through CSS extensions	no	no
Embedded HTML support	yes (using HTML 4.0 URI as namespace)	yes (using HTML 4.0 URI as namespace, some problems with HTML `img` elements)	yes (using `html` prefix, some problems with HTML `a` elements)	yes (using `html` prefix)
XSLT support	no (in development)	no (no announcements)	yes, based on old draft (new processor in development)	yes, based on old draft (new processor in development)
Release status	Preview Release	Beta	Shipping	Preview Release

Figure : **Browser XML Display Support Chart**

APPENDIX C :SOFTWARE CATALOGUE

Name	**Type of software**	**License info**	**Resource (URL)**
SAXON	DTD Generator XSLT processor	Mozilla Public License Version 1.0	http://users.iclway.co.uk/mhkay/saxon/index.html License: http://users.iclway.co.uk/mhkay/saxon/conditions.html
Front-end	DTD Generator	on-line	http://www.pault.com/Xmltube/dtdgen.html
MS Internet Explorer 5.5	Browser	Free (copy-righted)	http://www.microsoft.com/windows/ie/default.htm
Mozilla	Browser	open-source	http://www.mozilla.org
Netscape Preview Release 6	Browser	Free (copy-righted)	http://www.netscape.com
Opera 4.0	Browser	Commercial (free 30-day evaluation period)	http://www.opera.no/
DocZilla SGML/XML module alpha	Browser	Not free (Evaluation only)	http://www.doczilla.com
InDelv	Browser	open-source	http://www.indelv.com
SAX	Parser	Free for non-commercial and commercial use	http://www.megginson.com/SAX/
Lark 1.0	Non-validating XML processor	public use	http://www.textuality.com/Lark/
Larval 0.8	Validating XML processor	public use	http://www.textuality.com/Lark/
Expat	XML Parser Toolkit	free	ftp://ftp.jclark.com/pub/xml/expat.zip.
Microsoft XMLINT	XML Validation Tool	Freeware (Copyright Microsoft)	http://msdn.microsoft.com/xml/tools/xmlint.asp

Microsoft XML Notepad 1.5	XML editor	Freeware (Copyright Microsoft)	http://msdn.microsoft.com/xml/notepad/intro.asp
XMLwriter	XML editor	Commercial (free evaluation period)	http://xmlwriter.net/index.html
RSS File	RSSValidator Tool	On-line	http://my.netscape.com/publish/help/validate.tmpl
Site Summary	RSS editor	Free	http://www.zope.org/Members/edmundd/SiteSummary/
RSS Channel Editor	RSS editor	Free GNU	http://www.webreference.com/perl/tools/rssedit.zip
RSS Maker	RSS editor	Free GNU	http://www.webreference.com/perl/tools/makerss.pl
XML::RSS 0.8.	RSS editor	Free	http://www.perlxml.com/modules/XML-RSS-0.8.tar.gz Tutorial: http://www.webreference.com/perl/tutorial/8/
RSS2HTML	Converter from RSS to HTML	Free GNU	http://www.webreference.com/perl/rss2html.pl
Carmen's Headline	Viewer of RSS	Shareware	http://www.vertexdev.com/HeadlineViewer/
RDF for XML	RDF Build and Search Application	Free	http://www.alphaworks.ibm.com/aw.nsf/techmain/rdfforxml
XT	XSLT processor	Free	http://www.jclark.com/Xml/General XML/xt.html
4XSLT	XSLT processor	Free	http://fourthought.com/4Suite/4XSLT
iXSLT	XSLT processor	Free Trial	www.infoteria.com/en/contents/download/index.html
LotusXSL	XSLT processor	Commercial License	http://www.alphaworks.ibm.com/tech/LotusXSL
Transformiix	XSLT processor	open-source	http://lxr.mozilla.org/mozilla/source/extensions/transformiix
Resin	XSLT processor	open-source	http://www.caucho.com/products/resin/index.html
Xalan	XSLT processor	Free	http://xml.apache.org/xalan/overview.html
FOP	XSL-FO to PDF converter	Free	http://xml.apache.org/fop
FOP2PDF	XSL-FO to PDF converter	Commercial (free evaluation)	http://www.renderx.com/FO2PDF.html

Passive TeX	XSL-FO processor	free	http://users.ox.ac.uk/~rahtz/passivetex/
REXP	XSL-FO processor	open-source	http://www.esng.dibe.unige.it/REXP
EXcelon-Stylus	XSL-Enabled Authoring Tool	Commercial	http://www.exceloncorp.com/products/excelon-stylus.html
IBM XSL Editor	XSL-Enabled Authoring Tool	Commercial	http://www.alphaworks.ibm.com/tech/xsleditor
GMD-IPSI	XQL Engine	Free for non-commercial use and evaluation	http://xml.darmstadt.gmd.de/xql/index.html
XML::XQL	Perl extension to perform XQL queries	Free	http://www.perlxml.com/modules/
XML Query Engine Alpha version	XML Text Search Engine	Free for Evaluation and testing	http://www.fatdog.com/
Xdex	XML Text Search Engine	Commercial	http://www.xmlindex.com/
XML Similarity Search Engine	XML Text Search Engine	Commercial	http://www.infoglide.com/technology/techprod.htm
SIM	XML Text Search Engine	Commercial	http://www.simdb.com/
GoXML Search Engine	XML Text Search Engine	On-line	http://www.goxml.com/
fxgrep	XML Structured Query Engine	Source-code	http://www.informatik.uni-trier.de/Eneumann/Fxgrep
WebMethods B2B QueryView	XQL engine	Commercial	http://xml.webmethods.com/products/QueryView/
UC Berkeley's Cheshire	XML Structured Query Engine	On-line	http://cheshire.lib.berkeley.edu/
Xtenit	XML Structured Query Engine	Commercial	http://www.xtenit.com/
YAT	data conversion tool	Prototype	http://www-rocq.inria.fr/verso/Jerome.Simeon/YAT/

sgrep	grep searching	Free	http://www.cs.helsinki.fi/~jjaakkol/sgrep.html
NetObjects Collage	Web content management platform	Commercial	http://www.netobjects.com/
Redix	Family of XML/DTD/EDI products	Commercial	http://www.redix.com/
Dynabase 4.0	Web Content Management and Delivery System	Commercial	eBusiness Technologies Web Site: http://www.ebt.com/
Microsoft BizTalk Server 2000	Document Content Management System	Commercial	http://www.microsoft.com/biztalkserver/
Topic map	XML-enabled index of free XML tools	On-line	http://www.stud.ifi.uio.no/~lmariusg/linker/XMLtools.html http://www.garshol.priv.no/download/xmltools/xmltools-tm.xml
XML Database Products	List of products for Databases		http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLDatabaseProds.htm

GLOSSARY

API : Application Program Interface

B2B : Business- to- Business

CASE : Computer Aided Software Engineering

CDF : Channel Data Format

CDIF : CASE Data Interchange Format

CML : Chemical Mark-up Language

CSS : Cascading Style Sheet

CVS : Concurrent Versions System

DASL : DAV Searching and Locating

DAV : Distributed Authoring and Versioning

DOM : Document Object Model

DSSSL : Document Style Semantics and Specification Language

DTD : Document Type Declaration

EDI : Electronic Data Interchange

ERP : Enterprise Resource Planning

FO : Formatting Object

FOSL : Formatting Object Style sheet Language

HTML : HyperText Mark-up Language

HTTP : HyperText Transfer Protocol

IETF : Internet Engineering Task Force

JDBC : Java DataBase Connectivity

MathML : Mathematical Mark-up Language

MCF : Meta Content Framework

NGWS : New Generation Windows Services

OCS : Open Content Syndication

OMG : Object Management Group

OSD : Open Software Description

PDA : Personal Digital Assistant

PDF : Post-script Data Format

PI : Processing Instruction

PICS : Platform for Internet Content Selection

RCS : Revision Control System

RDBMS : Relational DataBase Management System

RDF : Resource Description Format

RPC : Remote Procedure Call

RSS : Rich Site Summary

SAX : Simple API for XML

SCCS : Software versioning and revision Control System

SCM : Scheme implementation

SDML : Signed Document Mark-up Language

SGML : Standard Generalized Mark-up Language

SMIL : Synchronized Multimedia Integration Language

SML : SmartX Mark-up Language

SQL : Structured Query Language

SSH : Secure SHell

SSL : Secure Socket Layer

SVG : Scalable Vector Graphics

TEI : Text Encoding Initiative

UML : Unified Modelling Language

URI : Uniform Resource Identifier

URL : Uniform Resource Locator

URN : Uniform Resource Number

VAN : Value Added Network

W3C : World Wide Web Consortium

WEBDAV : Web Distributed Authoring and Versioning

WIDL : Web Interface Definition Language

WAP : Wireless Application Protocol

WML : Wireless Mark-up Language

XHTML : eXtensible HyperText Mark-up Language

XLF : eXtensible Log Format

XLL : XML Linking Language

XMI : XML Metadata Interchange

XML : eXtensible Mark-up Language

XQL : XML Query Language

XSL : eXtensible Style sheet Language

XSLT : eXtensible Style sheet Language Transformations

LIST OF TABLES

LIST OF FIGURES

REFERENCES

[BOS97] BOSAK J. **:** XML, Java, and the future of the Web. Database interchange – Distributed processing – Web agents. (1997)

URL : http://sunsite.unc.edu/pub/sun-info/standards/xml/why/xmlapps.html

[BOUR99] BOURRET R. : XML And Databases. Discusses several issues and solutions for the exchange of data between an XML document and a database.(1999)

URL : http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm

[BRAY98] BRAY T. : XML.com's technical editor Tim Bray describes the significance of metadata and the relationship between RDF and XML. (1998)

URL : http://www.xml.com/xml/pub/98/06/rdf.html

[BUCK99] BUCK L. : Modeling Relational Data in XML (1999)

URL : http://www.extensibility.com/xml_resources/modeling.htm

[COVE98] COVER R. : XML and Semantic Transparency. (1998)

URL : http://www.oasis-open.org/cover/xmlAndSemantics.html

[COVE00] COVER R. : The XML Cover Pages about WebDAV (Extensions for Distributed Authoring and Versioning on the World Wide Web) (2000)

URL : http://www.oasis-open.org/cover/webdav.html

[DRAF00] XML Query Requirements W3C Working Draft (2000)

URL : http://www.w3.org/TR/2000/WD-xmlquery-req-20000131

[EISE00] EISENZOPF J. : Making Headlines with RSS - Using Rich Site Summaries To Draw New Visitors (2000)

URL : http://www.webtechniques.com/archives/2000/02/eisenzopf/

[GUID98] Guidelines for using XML for Electronic Data Interchange Version 0.05 (1998)

URL : http://www.geocities.com/WallStreet/Floor/5815/guide.htm

[HOGA97] HOGAN M. **:** XML The Foundation for the Future - XML-Enabled Technologies - XML Repository Requirements (1997)

URL : http://www.oasis-open.org/html/xml_foundation_future.html

[IDRI99] IDRIS N. : Benefits of using XML (1999)

URL : http://www.developerlife.com/xmlbenefits/default.htm

[IDRI99] IDRIS N. : Introduction to Sax and DOM APIs

URL : http://www.developerlife.com/saxvsdom/default.htm

[JOHN00] JOHNSON M. : XML for the absolute beginner - A guided tour from HTML to processing XML with Java (2000)

URL : http://www.javaworld.com/jw-04-1999/jw-04-xml.html

[IETF98] The Internet-Draft of Internet Engineering Task Force (IETF) on WebDAV Protocol 09 (1998)

URL : http://www.ics.uci.edu/pub/ietf/webdav/protocol/draft-ietf-webdav-protocol-09.txt

[KAM00] KAMTHAN P. **:** XML Euphoria in Perspective. Discusses the current state of XML and related initiatives, separating possibilities from popularizations. (2000)

URL : http://www.irt.org/articles/

[LADD00] LADDAD R. : XML APIs for databases. Presents a way to blend the power of XML and databases using custom SAX and DOM APIs. (2000)

URL: http://www.javaworld.com/javaworld/jw-01-2000/jw-01-dbxml.html

[LASS97] LASSILA O. : Introduction to RDF Metadata W3C NOTE (1997)

URL : http://www.w3.org/TR/NOTE-rdf-simple-intro-971113.html

[LATT99] LATTIG M. : XML stumbling blocks. Discusses the challenge of adopting XML for E-commerce companies. (1999)

URL : http://www.infoworld.com/cgi-bin/displayStory.p?/features/991122xml.htm

[LAUR00] LAURENT S. : A series of articles examining XML support in the Mozilla, Opera, and Internet Explorer browsers.(2000)

URL : http://www.xml.com/pub/au/St._Laurent_Simon

[MALE98] MALER E. : An Introduction to XML Linking (1998)

URL : http://www.xml.com/xml/pub/98/06/xlink/av.html

[MEGG00] SAX 1.0: The Simple API for event-based XML parsing. (2000)

URL : http://www.megginson.com/SAX/ **[MICR]** Microsoft XML Tutorial

URL : http://msdn.microsoft.com/xml/tutorial/default.asp

[MELN00] MELNIK S. : Database schemas for storing RDF (2000)

URL : http://www-db.stanford.edu/~melnik/rdf/db.htm

[MICR] Microsoft XML Tutorial

http://msdn.microsoft.com/xml/tutorial/default.asp

[MIKO98] MATSUMURA MIKO **:** XML speeds along in standards land. **(**1998)

URL : http://www.javaworld.com/jw-02-1998/jw-02-miko.html

[NETS99] RSS Quick Start Guide by Netcenter for creating one's own My Netscape channel(1999)

URL : http://my.netscape.com/publish/help/mnn20/quickstart.html

[ORAC] XML Support in Oracle

URL : http://technet.oracle.com/tech/xml/

XML Support in Jdeveloper 3.1

URL : http://technet.oracle.com/files/search/search.htm?Jdeveloper

XML Utilities for PL/SQL

URL : http://technet.oracle.com/tech/xml/info/index2.htm?Info&plsxml/xml4plsql.htm

[PHIP99] PHIPPS S. : IBM's chief XML evangelist examines how XML will change e-commerce. An interview to Infoworld (1999)

URL : http://infoworld.com/cgi-bin/displayStory.pl?interviews/991122phipps.htm

[REC98] Extensible Mark-up Language (XML) 1.0 W3C Recommendation (1998)

URL : http://www.w3.org/TR/REC-xml

[ROBI99] ROBIE J. : The Design of **XQL**, Texcel Research (1999)

URL : http://www.texcel.no/whitepapers/xql-design.html

[ROBI99] ROBIE J. : **XQL** Tutorial – A quick overview of XQL. Discusses the simplest XQL queries, which are also likely to be the most common. (1999)

URL : http://metalab.unc.edu/xql/xql-tutorial.html

[ROBI99] ROBIE J. : **XQL** Frequently Asked Questions (1999)

URL : http://metalab.unc.edu/xql/

[ROSE99] ROSENFELD L. : XML Text and Context (1999)

URL : http://webreview.com/wr/pub/1999/02/05/feature/index.html

[SEAR00] XML And Search – How Text Search Relates to XML Query Languages (1999-2000)

URL : http://www.searchtools.com/related/xml.html

[STEI00] STEIN G. : WEBDAV Frequently Asked Questions (2000)

URL : http://www.webdav.org/other/faq.html

[SUSA97] SUSAC D. **:** Artificial Intelligence & XML (Advantages-Syntax-Applications) (1997)

URL : http://ai.about.com/compute/ai/library/

[SWIC99] SWICK R. : Frequently Asked Questions about RDF (1999)

URL : http://www.w3.org/RDF/FAQ

[UCC99] Frequently Asked Questions about the Extensible Mark-up Language v. 1.5 (1999)

URL : http://www.ucc.ie/xml/faq.html

[WHIT99] WHITE C. : An Introduction to XSL (1999)

URL : http://www.javertising.com/webtech/xml.htm

[WHIT99] WHITEHEAD J. : The Future of Distributed Software Development on the Internet : From CVS to WebDAV to Delta-V. A Review (1999)

URL : http://www.webtechniques.com/archives/1999/10/whitehead/

[WIGG99] WIGGIN P. : RSS Delivers the XML Promise - Why Would You Use RSS? (1999)

URL : http://webreview.com/pub/1999/10/29/feature/index2a.html

[WONG00] WONG W. : CNET News.com - Microsoft to take wraps off new Windows strategy (2000)

URL : http://news.cnet.com/news/0-1003-200-2122784.html?tag=st.ne.1002.tgif.ni

[XML/EDI]XML + EDI : The e-Business Framework: Definitions , Links, Comments

URL : http://www.geocities.com/WallStreet/Floor/5815/

Some more interesting links:

WebDAV Resources – A central resource for documentation, specifications, software, mailing lists, links for WebDAV

URL : http://www.webdav.org/

XML Searching Resources – XML Text Search Engines – XML Structured Query Engines – XML Query Languages

URL : http://www.searchtools.com/related/xml-resources.html

XSL Extensible Style sheet Language by The XML Cover Pages. Introduction to XSL – Specifications — XSL/XSLT Articles, Papers, Tutorials – Sample XSL Style sheets (March 01, 2000)

URL : http://www.oasis-open.org/cover/xsl.html

Microsoft XSL Developer's Guide

URL : http://msdn.microsoft.com/xml/xslguide/

XSL Concepts and Practical Use

URL : http://www.nwalsh.com/docs/tutorials/webtek2000/xsl/

The XML Cover Pages about Extensible Mark-up Language (XML) (March 29, 2000)

URL : http://www.oasis-open.org/cover/xml.html

XML Resources – Links – Tools

URL : http://www.jclark.com/xml

A page with many useful links for XML

URL : http://www.geocities.com/SiliconValley/Peaks/5957/xml.htm