a study of

eXtensible Markup Language

(XML)

Author: Pontus Norman d94-pno@nada.kth.se
Date: 25 February 1999

Abstract

The attention paid by the Internet community to Extensible Markup Language (XML) is impressive. XML has been heralded as the next important Internet technology, the next step following the HyperText Markup Language (HTML), and the natural and worthy companion to the Java programming language.

HTML defines a single, fixed type of document with markups that let you describe a common class of simple office-style reports. As Web documents have become larger and more complex, Web content providers have begun to experience the limitations of a medium that does not provide the extensibility, structure, and data checking needed for large-scale commercial applications. The ability of Java applets to embed powerful data manipulation capabilities in Web clients makes even clearer the limitations of current methods for the transmittal of document data.

XML takes document markup to the next level, offering a human-readable, self-explaining, well-structured, extendable and consistent way to describe and transfer data. XML promises to be as big a revolution to the Web as HTML was.

This paper describes the XML effort, makes a survey of most of the associated specifications and discusses new kinds of Java-based Web applications made possible by XML. The paper also discusses the impact that XML will have on some of the existing technologies, like Electronic Data Interchange (EDI) and the exchange of industrial data (STEP). The end of this paper describes in detail one of the first efforts made to use XML in an industrial application.

Table of Contents

A STUDY OF EXTENSIBLE MARKUP LANGUAGE (XML)

ABSTRACT 2
TABLE OF CONTENTS 3
ABOUT THIS DOCUMENT 6
INTRODUCTION 7

PART 1: INTRODUCTION TO XML 8

1.1 WHERE HAS XML COME FROM? 8
1.1.1 World Wide Web Consortium (W3C) 8
1.1.2 XML Background 8
1.2 WHY XML? 8
1.3 BASIC XML 10
Example: Car register in HTML 10
Example: Car register in XML 10
1.3.1 Elements 11
1.3.1.1 Attributes 11
Example: An element with attributes 12
1.3.1.2 Elements versus Attributes 12
1.3.2 Entities 12
Examples: Internal entities 12
Examples: External entities 13
1.3.3 Document Type Declaration (DTD) 14
Example: A DTD 14
1.3.3.1 Valid or Well-Structured? 14
Example: A well-structured XML document 14
Example: A valid and well-structured XML document. 15
1.3.3.2 DTDs and Modularity 15
Example: Combining several DTDs into one document 15
1.3.4 XML and Datatypes 15
More complex datatypes 16
Example: A binary tree in XML 16
1.4 XML AND JAVA – FRIENDS OR ENEMIES? 17

PART 2: RELATED SPECIFICATIONS 19

2.1 W3C SPECIFICATIONS AND LEVELS 19
2.2 DOCUMENT OBJECT MODEL (DOM) 19
2.3 STYLESHEET LANGUAGES 20
2.4 CASCADING STYLE SHEETS (CSS) 21
2.4.1 Cascading Style Sheets Level 2 (CSS2) 21
2.4.2 Viewing XML using CSS2 (An Example) 21
2.4.2.1 Creating the CSS2 stylesheet 22
2.4.2.2 Associating the CSS stylesheet 22
2.5 EXTENSIBLE STYLESHEET LANGUAGE (XSL) 22
2.5.1 Origin 23
2.5.2 How does XSL work? 23
2.5.2.1 Patterns 24
2.5.2.2 Actions 25
2.5.2.3 Flow Objects 25
Microsoft – XSL and CSS 25
2.6 EXTENSIBLE LINKING LANGUAGE (XLL) 26
2.6.1 Origin 26
2.6.2 XLink 26
2.6.2.1 How does XLink work? 27
2.6.3 XPointer 27
2.7 XML – NAMESPACES 28
2.7.1 Using Namespaces – XML Syntax 28
Example 1: 28
Example 2: 29
Example 3: 29

Example 4: 29
2.8 XML-DATA 30
Example: Using the dt:dt attribute 30
2.8.1 Current Support *30*
Data Type 30
2.9 XML QUERY LANGUAGE (XML-QL) 31
2.9.1 Using XML-QL *32*
Example: An XML-QL query 32
2.10 SCHEMA FOR OBJECT-ORIENTED XML (SOX) 32
2.10.1 Goals and Requirements *33*
2.10.2 Features *34*
2.10.2.1 Base element types 34
2.10.2.2 Datatypes 34
2.10.2.3 Documentation 34
2.10.2.4 Inheritance 34
2.10.2.5 Namespace support 34
2.10.2.6 XML syntax and validation 34
2.10.3 SOX – A Detailed Example *35*
Annotated Example 35
2.11 VECTOR MARKUP LANGUAGE (VML) 36
2.11.1 Requirements *37*
2.11.2 Structure *37*
2.11.3 Use of CSS *38*
2.11.4 Future Support *38*

PART 3: USING XML 39

3.1 A COMMON XML COMMUNICATION ARCHITECTURE 39
3.1.1 Three-Tier Application Architecture *39*
3.2 SUPPORT FOR XML (AND ASSOCIATED STANDARDS) 40
3.2.1 XML Parsers *41*
Lightweight 41
3.2.2 Future Support *41*
3.3 AN EXAMPLE OF XML AND XSL 43
3.4 XML AND ELECTRONIC TRANSACTIONS 43
3.4.1 EDI *44*
3.4.2 EDI using XML *44*
3.4.2.1 EDI using XML – Example 45
Listing 1: Plain purchase order 45
Listing 2: Fragment of ANSI X12 transaction set corresponding to Listing 1 46
Listing 3: XML document analog to the X12 transaction set in Listing 2 46
3.4.2.2 XML-EDI Tags 47
3.4.2.3 Repositories 47
Example: An XML-EDI transaction using a global repository 48
3.4.2.4 Current and future support 48
3.5 PRODUCT DATA 49
3.5.1 STEP *49*
3.5.1.1 EXPRESS 50
Example: Car data as EXPRESS code 50
Example: Car data as an EXPRESS-G graphical representation 50
3.5.1.2 Data Models 50
3.5.1.3 Implementation Forms 51
Example: A car data instance, stored using STEP Part 21 51
3.5.2 STEP or XML? *51*
3.5.2.1 Conclusion 52
3.5.3 STEP and XML? *52*
3.5.4 STEP or SOX? *52*

PART 4: TRYING OUT XML 54

4.1 THE APPLICATION – PASTILL 54
4.1.1 What should Pastill do? *54*
4.1.1.1 Functionality 54
4.1.1.2 Other requirements 54
4.1.2 Design *55*
4.1.2.1 Database 55

4.1.2.2 Server 56
4.1.2.3 Client 56
4.1.2.4 Where does XML fit? 56
4.1.3 How Pastill works - Flowcharts *57*
4.1.3.1 The Client Requests Data 57
4.1.3.1.1 Client Transaction 57
4.1.3.1.2 Server Transaction 58
4.1.3.2 The Client wants to store data 59
4.1.3.2.1 Client Transaction 59
4.1.3.2.2 Server Transaction 60
4.1.4 At the Heart of Pastill – an XML Parser *61*
4.1.5 Walkthrough of Pastill *61*
4.1.5.1 Layout 61
4.1.5.2 Entering Pastill 62
4.1.5.3 Retrieving Journals 63
4.1.5.4 Working with Journals 63
4.1.5.5 Working with patient visits 64
4.1.6 Experiences learned by using XML in Pastill *64*
4.1.6.1 Advantages of using XML 64
4.1.6.1.1 Very easy communication 64
4.1.6.1.2 Bandwidth Usage 64
4.1.6.1.3 No need to have a real-time connection with the Server 65
4.1.6.2 Problems with using XML 65
4.1.6.2.1 It's hard to write a good structure 65
4.1.6.2.3 Tags take up lots of space 65
4.1.6.2.4 Too much string handling slows down the application 66
4.1.2.6.5 Problems with the XML4J Parser - Bugs 66
4.1.3 Pastill as a project *66*
4.1.3.1 Working in a team 67
4.1.3.2 Time planning 67
4.1.4 Summary – Pastill *67*
4.2 FUTURE WORK 68
4.2.1 New Specifications pop-up all the time 68
4.2.2 Continued follow-up as Specifications evolve 68
4.2.3 Continued follow-up of software companies' XML efforts 68
4.2.4 Continued follow-up of XML-EDI effort 68
4.2.5 Continued follow-up of XML-STEP effort 68
4.2.6 Trying out more Specifications 68
4.3 SUMMARY 68

PART 5: ASSORTED INFORMATION 70

5.1 ABBREVIATIONS 70
5.2 BIBLIOGRAPHY 71

About this document

This thesis is a part of the requirements for my (Pontus Norman's) Master of Science degree in Computer Science at the Department of Teleinformatics, Royal Institute of Technology in Sweden. The work behind this document was conducted at Decerno AB in Sweden.

Since the specifications that this document is based on are still subject to change, the statements made in this document are only valid for the specification versions specified in the bibliography. I take no responsibility about the correctness of this document as the specifications evolve.

I would like to take this opportunity to thank my supervisors at Decerno - Leif Pettersson and Wilhelm Arnör, my fellow programmers in the Pastill project - Leif Pettersson (again) and Daniel Lekberg, as well as my supervisor and examiner at the Royal Institute of Technology, Prof. Gerald Q. Maguire Jr, for their help and support with this work.

In my thesis, excerpts from the following World Wide Web Consortium documents are included in accordance with the W3C IPR Document Notice, http://www.w3.org/Consortium/Legal/copyright-documents.html. Copyright © World Wide Web Consortium (Massachusetts Institute of Technology, Institut National de Recherche en Informatique et en Automatique, Keio University). All Rights Reserved.

Extensible Markup Language (XML) Version 1.0, *World Wide Web Consortium Recommendation 10-February-1998*
http://www.w3.org/TR/1998/REC-XML-19980210

Cascading Style Sheets Level 1 (CSS1), *World Wide Web Consortium Recommendation 17-December-1996*
http://www.w3.org/TR/REC-CSS1

Cascading Style Sheets Level 2 (CSS2), *World Wide Web Consortium Recommendation 12-May-1998*
http://www.w3.org/TR/1998/REC-CSS2-19980512/

Namespaces in XML, *World Wide Web Consortium Recommendation 14-January-1999*
http://www.w3.org/TR/1999/REC-xml-names-19990114

Extensible Stylesheet Language (XSL) Version 1.0, *World Wide Web Consortium Working Draft 18-August-1998*
http://www.w3.org/TR/1998/WD-xsl-19980818

Extensible Linking Language (XLink), *World Wide Web Consortium Working Draft 3-March-1998*
http://www.w3.org/TR/1998/WD-xlink-19980303

XML Pointer Language (XPointer), *World Wide Web Consortium Working Draft 3-March-1998*
http://www.w3.org/TR/1998/WD-xptr-19980303

Document Object Model (DOM) Level 1 Specification Version 1.0, *World Wide Web Consortium Recommendation 1-October-1998*
http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/

Schema for Object-oriented XML (SOX), *World Wide Web Consortium Note 15-September-1998*
http://www.w3.org/TR/1998/NOTE-SOX-19980930

XML-QL: A query language for XML, *World Wide Web Consortium Note 19-August-1998*
http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

Vector Markup Language (VML), *World Wide Web Consortium Note 13-May-1998*
http://www.w3.org/TR/1998/NOTE-VML-19980513

Pontus Norman
February 1999
Decerno AB, Näsby Park, Sweden

Introduction

The World Wide Web (WWW) is today an enormous market and it will continue to grow in the future as more and more people are being connected to it. This global market has attracted companies and Web publishers to create applications and Web pages in ever-larger numbers. This in turn has led to a growing development of standards and structures to assist their efforts. The developers are being overwhelmed by the continuous flow of new standards that promises much but do they deliver?

Decerno AB is a Swedish company that has dedicated much of its recent work to creating interactive applications on the Web. One of the company's strongest fields is in the development of business supporting, administrative systems based on database solutions and with a user interface on the Internet. The structure of these systems is quite typical of many of the applications being developed around the world today.

- A client, that is downloaded to the user's computer over the Internet, present the graphical interface of the application.
- A server located somewhere in the Internet provides the client with the data that it requests.
- A database to store the data used by the application.

The key that makes these applications work is communication. Communication is needed between the client and the server; communication is also needed between the server and the database. Another very important part of any application is the Graphical User Interface (GUI), "*to the user the GUI **is** the application*" as the saying goes.

The need for good communication is what makes or breaks many applications in development. For many years the only standard that was publicly available was the HyperText Markup Language (HTML) [13]. Now there is a multitude of other standards: Dynamic HTML (DHTML) [38], Common Object Request Broker Architecture (CORBA) [39], DCOM (Distributed Component Object Model) [37,41] and Active Server Pages (ASP) [50], to mention only a few. The problem facing developers today are what to choose; what are the limitations, possibilities, advantages and disadvantages of the different technologies. So therefore there is a need to understand and have knowledge of as many such architectures as possible.

One such new and exciting standard in development right now is the World Wide Web Consortium's (W3C's) Extensible Markup Language (XML). This standard promises to be THE standard for Web development and structured data transfer. This paper is a thorough examination of this standard to determine if it keeps its promises.

This study of XML consists of four parts.

1. An introduction of XML, including syntax and examples.
2. A brief examination of the associated standards as they are described in their drafts and in the available literature.
3. An examination of what uses XML will have in reality and what support is currently available.
4. An implementation and evaluation of XML in one of the applications (Pastill) currently under construction at Decerno.

The general questions that Decerno wants answered are:

- What is XML?
- What can XML be used for?
- Does XML have a future?

And more specifically:

- In Internet applications there can be problems with maintaining real-time connections with the server (the connection may not always exist). Does XML in any way provide an alternative to this with, for instance, necessity-driven delayed communication with the server? This means holding off any client – server communication until a connection can be established and meanwhile cashing all transactions on the client.
- Communication between applications is often handled using different file-based standards for the syntax. By file-based communications I mean that one application creates a file with a certain format and transfers it to another application that interprets the file. For example, this procedure is used for Electronic Data Interchange (EDI) and the file based-standards used are everything from homemade in-house formats to application standards such as the ANSI X12 format [32] and EDIFACT [30]. Can XML be a concrete alternative to these and in what extent?

Part 1: Introduction to XML

1.1 Where has XML come from?

XML stands for Extensible Markup Language [1], and is defined by the World Wide Web Consortium (W3C) [http://www.w3.org], the same people who has constructed the more well known HyperText Markup Language (HTML) [13].

1.1.1 World Wide Web Consortium (W3C)

W3C had at last count 275 member organizations, including companies, nonprofit organizations, industry groups and government agencies from all over the world [http://www.w3c.org/Consortium/Member/List.html]. This power-packed assemblage is the closest thing the famously decentralized Web has to a governing body. More than the U.S. government, whose funding created the Internet, and more than the telephone companies whose wires and fibers carry the net's digital traffic, it is the W3C that will largely determine the Web's structure in the 21st century.

For a group with this much clout, W3C isn't well known. Nor does it court fame. Its meetings are closed to outsiders. Although the consortium, based at MIT, has brought some order to the unruly thickets of the Web, it has plenty of critics who say the group has become a significant maker of public policy - and ought to start acting like one. They argue that the W3C should open its membership and meetings to broader, more democratic participation.

The W3C is not a standards organization in the mold of such traditional outfits as the American National Standards Institute (ANSI) or the International Standards Organization (ISO). Think, instead, of W3C as a group of technologists who issues recommendations. Legally W3C recommendations have no teeth; even consortium members are under no obligation to implement them. In practice, however, W3C's recommendations carry a moral authority that is the closest thing the Internet has to law. Microsoft, Netscape and a host of other companies have pledged to implement the standards in their products. And this moral authority has given rise to the W3C's technical work - which is almost universally praised - as well as its policy-making activities, which have generated considerable controversy.

1.1.2 XML Background

XML is a subset of the Standard Generalized Markup Language (SGML) [43]. SGML is a way to express structure and content in different types of electronic documents, it has been around for more than a decade. A better known and more limited sibling to XML is HTML. Today HTML is very popular and is used for creating Web pages that can be viewed all around the world. Because these two languages are related they share some of the same characteristics such as similar syntax and the usage of tags inside of brackets. One important difference is however the fact that HTML is an application format based on SGML, unlike XML which is a subset of SGML. This distinction between the two is important because HTML can not be used to define new applications as easily as XML can.

Since XML is a subset of SGML, applications that can read SGML can also read XML, however the opposite is not true. One of the reasons SGML has never reached widespread public usage is that it has been considered to be too complex. XML was developed to be less complex than SGML and to be able to work in limited bandwidth networks such as the Internet. W3C predicts that HTML, SGML and XML will all be used in the future, none makes the others obsolete. HTML will continue to be the easiest and fastest way to publish information on the Web. More advanced Web pages and applications will increasingly use XML. SGML will live on in high-end, highly structured publishing applications.

It should be noted that XML and associated specifications are still under development and new drafts are still being posted [http://www.w3.org/TR/]. In other words it is not certain that this paper is 100% correct about all the details in the different specifications since these are still subject to change. It should also be noted that new specifications based on XML are popping up almost every month.

1.2 Why XML?

The remarkable growth of the WWW in recent years has been enabled by the possibility to cheaply and easily distribute information and applications to users all over the world. As the information on the Web

gets more complex and the number of users keeps growing, more people become aware of the limitations in current technologies and standards. Add to that the growing popularity of Java clients with powerful data processing capabilities and the absence of suitable standards to transfer structured data is obvious.

The limitations when constructing Web pages stems from the fact that HTML uses a fixed number of predefined tags to construct the appearance of a Web page. It is this inability to extend the functionality of the language that has made the development of a new standard for Web publishing desirable. Already some Web publishers have switched to using SGML for more advanced publications, but this standard is considered too large and complex to gain public acceptance and widespread usage.

Distributed applications in general and Java in particular has in recent years shown problems caused by the absence of a uniform standard to transfer structured data. Despite the fact that large efforts and huge amounts of money have been put into this area, no publicly approved and used standard has been developed. The problems occur when different applications want to talk to each other, because they almost always use their own incompatible in-house standards for communicating data.

XML was developed to solve these problems. It offers a structured and consistent way to describe and transfer data. The power and beauty of XML is that it maintains the separation of the user interface from the actual data. This distinction makes XML not only a language for creating advanced Web pages but it can also be used in a more general sense to transfer data. Applications using XML can read data from each other, without having prior knowledge of incoming data formats, because the data contained in the XML documents are self-describing.

W3C has set up some goals for the development of XML; here is a list of these goals and comments on them:

1. **XML shall be straightforwardly usable over the Internet.** Users must be able to view XML documents as quickly and easily as HTML documents. In practice, this will only be possible when XML browsers are as robust and widely available as HTML browsers, but the principle remains.
2. **XML shall support a wide variety of applications.** XML should be beneficial to a wide variety of diverse applications: authoring, browsing, content analysis, etc. Although the initial focus is on serving structured documents over the Web, it is not meant to narrowly define XML.
3. **XML shall be compatible with SGML.** Most of the people involved in the XML effort come from organizations that have a large, in some cases staggering, amount of material in SGML. 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 shall be easy to write programs which process XML documents.** The colloquial way of expressing this goal while the specification was being developed was that 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.** Optional features inevitably raise compatibility problems when users want to share documents and sometimes lead to confusion and frustration.
6. **XML documents should be human-legible and reasonably clear.** If you don't have an XML browser and you've received a hunk of XML from somewhere, you ought to be able to look at it in your favorite text editor and actually figure out what the content means.
7. **The XML design should be prepared quickly.** Standards efforts are notoriously slow. XML was needed immediately and was developed as quickly as possible.
8. **The design of XML shall be formal and concise.** In many ways a corollary to rule 4, it essentially means that XML must be expressed in Extended Bachus-Naur Notation (EBNF) and must be amenable to modern compiler tools and techniques. There are a number of technical reasons why the SGML grammar *cannot* be expressed in EBNF. Writing a proper SGML parser requires handling a variety of rarely used and difficult to parse language features. XML does not.
9. **XML documents shall be easy to create.** Although there will eventually be sophisticated editors to create and edit XML content, they will not appear immediately. In the interim, it must be possible to create XML documents in other ways: directly in a text editor, with simple shell and Perl scripts, etc.
10. **Terseness in XML markup is of minimal importance.** Several SGML language features were designed to minimize the amount of typing required to manually key in SGML documents. These features are not supported in XML. From an abstract point of view, these documents are indistinguishable from their more fully specified forms, but supporting these features adds a

considerable burden to the SGML parser (or the person writing it, anyway). In addition, most modern editors offer better facilities to define shortcuts when entering text.

It is clear that XML has a broad support among leading companies:

"I think XML is really a breakthrough, because it brings the database and the publishing world into having an abstract way of describing properties."
-- Bill Gates

Sun Microsystems Scientific Office director John Gage predicts that XML will emerge as the glue to bind electronic commerce applications and turn the computer into ***"an extensible linked document and database."***

1.3 Basic XML

W3C has called XML "*a common syntax for expressing structure in data*". To put structure in data means: structuring the document depending on the content, meaning, or usage of the data. XML documents contain character data and markup. The character data is often referred to simply as content, while the markup provides structures for that content. The distinction between data and markup is drawn more sharply and more visibly in XML than it is in many other systems.

The simplest way to describe the structure of XML is to compare it to HTML which most people are already familiar with. As in HTML, the structure in XML is built up using markup tags. There is however a very important difference between tags in HTML and tags in XML; unlike HTML tags, XML tags have no predefined meaning. In addition XML is extensible, structured, and can be validated. XML separates data and the presentation of data. This means that the same XML document can be displayed in a number of ways depending on the media of the presentation without changing the underlying structure of the data.

These two examples clearly show the similarities and differences between HTML and XML.

Example: Car register in HTML

```
<H1> Car Register </H1>
<H2> Registration number: ABC123 </H2>
<H2> Make: Saab 9000 </H2>
<H2> Model: 1995 </H2>
<H2> Owner: </H2>
<p>  Kalle Karlsson
     Götgatan 1
     11111 Stockholm
</p>
```

All tags say something about how the data will be displayed. No other information about structure or relation between the data is saved in the document. As a result little can be done with the data other than displaying it.

Example: Car register in XML

```
<Car Register>
       <Car>
              <Registration Number> ABC123 </Registration Number>
              <Make> Saab 9000 </Make>
              <Model> 1995 </Model>
              <Owner>
                     <Name> Kalle Karlsson </Name>
                     <Address> Götgatan 1 </Address>
                     <Zip code> 11111 </Zip code>
                     <City> Stockholm </City>
              </Owner>
       </Car>
</Car Register>
```

As can be seen none of the tags hold any information about the view of the data they contain. They only contain information about the structure and content of the document. The appearance of the data is left for the application reading the document to decide. In the example we clearly see that it is the markup tags that build up the formal structure of the document. The labels in the tags can be freely chosen by the author of the document, the only restriction is that each tag in some way should describe the data that is stored within

it. It should also be noted that unlike HTML tags, tag names in XML are case sensitive, which mean that `<Name>` is not the same as `<name>`.

1.3.1 Elements

As has been previously shown, the structure in XML is built up by markup tags and character data in groups that are normally called elements. Each element represents a logic component of the XML document. All elements consist of one of the following constructs:

- A tag together with the character data that the tag describes.
  Example: `<Make> Saab 9000 </Make>.`
- A tag together with other elements (which in turn can consist of other tags and elements and so on)
  Example:

```
<Owner>
      <Name> Kalle Karlsson </Name>
      <Address> Götgatan 1 </Address>
      <Zip code> 11111 </Zip code>
      <City> Stockholm </City>
</Owner>
```

Every XML document therefore consists of elements hanging together in a logical tree structure. There is always one element that contains all the other elements; this element is called the root element. The tree structure of the elements is a vital part of XML and is used when an application wants to read and interpret an XML document.

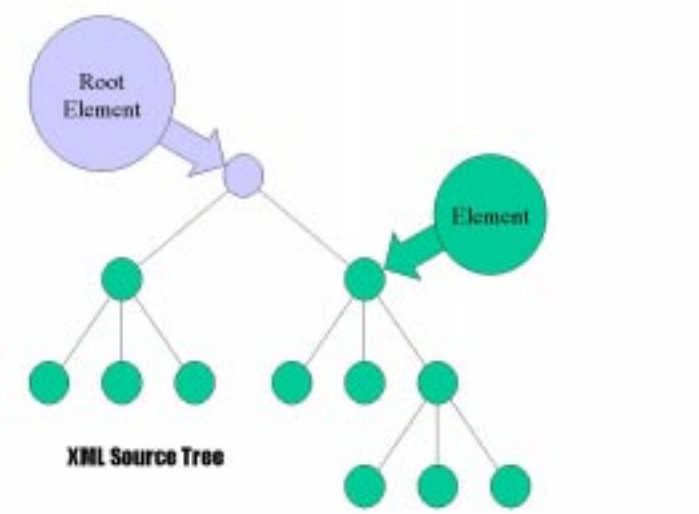


Figure 1: The structure of the XML source tree; the root node and the connected elements.

The values stored within the elements can basically be of three different types, parsed character data (PCDATA), unparsed character data (CDATA), or processing instructions (PI).

- **PCDATA –** Is used to store marked-up text that will be evaluated by the XML parser.
- **CDATA –** Is used to store marked-up text so that the markup is not evaluated, i.e. CDATA sections provide a way of making sure specific markup is not interpreted as markup. CDATA sections begin with the string `<![CDATA`, and end with the string `]]>`.
- **PI –** Holds processing directions and information passed to XML parsers and programs. PI instructions always start with `<?`, and end with `?>`. An example of a PI is the code at the beginning of every XML document that tells the parser which version of XML the document contains `<?XML version = “1.0” ?>`.

1.3.1.1 Attributes

Elements can also have other information attached to them called attributes. Attributes describe properties of elements. For instance it is sometimes useful to have an attribute containing the security demands on the person or application that wants to access that element. Attributes in XML look very similar to attributes in HTML.

Example: An element with attributes

```
<Employees>
       <!- The Person elements has three attributes, email, phone, and fax ->
       <Person email  = “any@where” phone = “111111” fax = “111112”>
              <Name> Kalle Andersson </Name>
       </Person>
       <Person email = “some@where” phone = “222222” fax = “222223”>
              <Name> Johan Olsson </Name>
       </Person>
</Employees>
```

1.3.1.2 Elements versus Attributes

One very interesting and much debated issue is when to save the data content in an element and when to save it as an attribute. The general principle seems to be that metadata (information about information) should be saved in attributes and the actual information should be stored in elements.

“Determine if the data in question is fundamentally metadata or content. Metadata is information that describes the container while content is the information the container conveys. For example, an ID attribute is clearly metadata because it describes the containing element (by giving it a unique) name. The author of a document is also metadata, as is the title (in my view). One way to distinguish metadata from content is to ask the question:
-If I removed this data, would my understanding of, or ability to comprehend, the content change? If the answer is no, it's metadata, if the answer is yes, it's content (or annotation, which is the third fundamental class of information). For example, knowing or not knowing the author of some information doesn't affect your ability to understand the content in normal practice (you can always think of weird cases where knowledge of the author is required, but these are games, not workaday information objects).”
-- W. Eliot Kimber, Senior Consulting SGML Engineer, ISOGEN International Corp.

1.3.2 Entities

An XML document can be defined as a linear series of characters and references to other objects. An XML processor starts at the beginning of the document and works down to the end. XML provides a mechanism for allowing the text and objects in the document to be organized non-linearly. The parser then reorganizes it to the linear structure. The mechanisms that make this possible are called entities. An entity can be as small as a single character or as large as an entire XML document. An XML document can be broken up into many files on a hard disk or objects in a database and each of them is called an entity in XML terminology. Entities can even be spread across the Internet. Whereas XML elements describe the XML documents logical structure, entities keep track of the location of the chunks of bytes that make up the document.

Very simplified an entity consists of a name and a content. The content is the actual stored data and the name is used to refer to that data. There are several different kinds of entities used for different purposes.

If an entity is defined without any separate storage file, and the content is given in its declaration, the entity is called an *internal entity*. All internal entities are parsed entities. This means that the XML processor parses them like any other XML text. The name parsed is somewhat badly chosen since the entities are in-fact unparsed until they are actually used.

Examples: Internal entities

- **Internal entities used as an abbreviation**
  `<!ENTITY dtd “document type definition”>`
  Whenever the parser comes across a reference (`%dtd`) to this entity it will replace that reference with the text: `document type definition.`
- **Internal entities with markup**
  `<!ENTITY dtd “<term>document type definition</term>”>`
  The internal entity can also contain markup.

External entities get their contents from somewhere else in the system. The location of the content is identified using an external identifier; usually this is just the world `SYSTEM` followed by a Uniform Resource Identifier (URI) [46,47,51]. External entities are either parsed or unparsed depending on their

content. Imagine the number of error messages that would occur if an XML processor tried to parse a graphic image as if it was made up of XML text!

Syntactically, declarations of unparsed entities are differentiated from those of other external entities by the keyword `NDATA` followed by a notation name.

Examples: External entities

- **External parsed entities**
  ```
  <!ENTITY external_chapter SYSTEM "http://www.book.com/chapter1.xml">
  ```
  The content of the entity named `external_chapter` will be the entire XML document `chapter1.xml`.
- **External unparsed entities**
  ```
  <!ENTITY front_page SYSTEM "http://www.book.com/frontpage.gif" NDATA GIF>
  ```
  The entity `front_page` will contain the GIF-picture `frontpage.gif`. The keyword `NDATA` shows that it is an unparsed entity with notation name `GIF`.

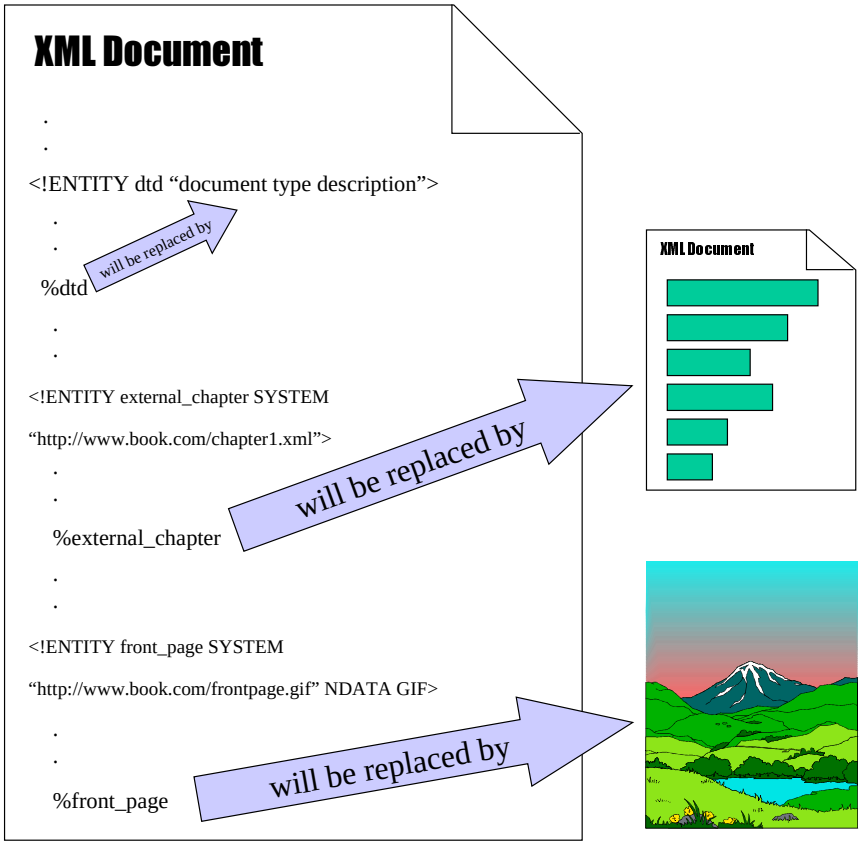


Figure 3: Shows how the entities will be replaced with their content by the XML parser.

There are many interesting things that can be done using entities.

- You can store every chapter of a book in a separate file and link them together as entities.
- You can insert often-used text, such as a product name, into an entity so that it is consistently spelled and displayed throughout the document. When the product name entity is updated to reflect a new version, the change would be instantly visible anywhere the entity was used.
- You can create an entity that would represent "legal boilerplate" text (such as software license) and reuse that entity in many different documents.
- You can integrate pictures and multimedia objects into the XML documents.
- You can develop "document type definition components" that can be used in many document type definitions. These would allow you to reuse the declaration for common element types across several documents.
- And much more…

1.3.3 Document Type Declaration (DTD)

An important part of XML is the ability to at the beginning of an XML document store information about what the rest of the document will contain; this information is called the Document Type Declaration (DTD). The DTD defines what markup tags can be used in the document, what order they can appear in, what other tags they can contain and so on. In XML it isn't strictly necessary to have a DTD associated with each XML document but it is considered bad manners not to have one. Also a DTD is necessary to be able to check if an XML document is valid and/or well-structured. These two terms will be defined below. The following example shows the DTD for the previous employee example.

Example: A DTD

The DTD consists of a number of `<!ELEMENT>` and `<!ATTLIST>` tags that are used to define the structure of the rest of the XML document. This is the DTD for the previous employee example.

```
<DOCTYPE Employees [
   <!ELEMENT Employees (Person)>
   <!ELEMENT Person(Name)>
   <!ATTLIST Person
      email PCDATA #IMPLIED
      phone PCDATA #IMPLIED
      fax   PCDATA #IMPLIED>
   <!ELEMENT Name #PCDATA>
]>
```

The DTD is used to define the model that the rest of the data in the document must follow. The DTD can be fetched from an external source or be imbedded in the XML document itself. Today there exist big efforts to develop standardized DTDs for different areas. One such standardization is the official DTD-specification for XML [26], in this specification there are standardized DTDs for lists, references, illustrations, and more.

1.3.3.1 Valid or Well-Structured?

In any language it is necessary to have a mechanism that checks if a written code conforms to the grammar and specifications of that language.

The earliest versions of HTML browsers validated, but there's this evil thing called "bad HTML" that quickly reared its ugly head. No HTML browser today can possibly afford to validate. Instead, they compete on how many violations of HTML syntax they can "add value to". This is another reason why XML is needed; to make one solution that works everywhere.

"I want to build one Web site and have it work everywhere. That's something we could do in 1994, and not today. That's not progress."
-- Glenn Davis, Web Standards Project.

To avoid such problem in XML, there are two such structural and grammatical checks: is an XML document valid and/or well-structured (also called well-formed in some documentation).

An XML document is considered **well-structured** IFF:

- it contains one or more elements;
- there is precisely one element (the root element) for which neither the start nor the end tag is inside any other element;
- all other tags must nest within each other correctly, and
- all entities used in the document must either be predefined in XML or in the DTD. (Think of an entity as a part of the data that makes up the XML document.)

An XML document is considered **valid** IFF there is a DTD associated with it, and the document complies with that DTD.

Example: A well-structured XML document

```
<?XML version = "1.0" ?>
<Name> Kalle Karlsson </Name>
```

That's all it takes to make a well-structured XML document.

Example: A valid and well-structured XML document.
To create a well-structured and valid XML document I simply add a DTD to the previous car register example.

```
<?XML version = "1.0" ?>
<DOCTYPE Car Register [
<!ELEMENT Car Register (Car)*>
<!ELEMENT Car (Registration Number, Make, Model, Owner)>
<!ELEMENT Registration Number (#PCDATA)>
<!ELEMENT Make (#PCDATA)>
<!ELEMENT Model (#PCDATA)>
<!ELEMENT Owner (Name, Address, Zip Code, City)>
<!ELEMENT Name (#PCDATA)>
<!ELEMENT Address (#PCDATA)>
<!ELEMENT Zip Code (#PCDATA)>
<!ELEMENT City (#PCDATA)>
]>

 <Car Register>
       <Car>
              <Registration Number> ABC123 </Registration Number>
              <Make> Saab 9000 </Make>
              <Model> 1995 </Model>
              <Owner>
                     <Name> Kalle Karlsson </Name>
                     <Address> Götgatan 1 </Address>
                     <Zip Code> 11111 </Zip Code>
                     <City> Stockholm </City>
              </Owner>
       </Car>
 </Car Register>
```

First in the XML document a DTD named `Car Register` is first defined. After follow the XML elements in a structure that follow that DTD.

1.3.3.2 DTDs and Modularity

XML provides the powerful possibility to combine several DTDs in a single XML document through the use of entities. This is important because it enables development using a modular approach. Developers will in the future be able to combine standardized, well-known and trusted DTDs to get the desired structure of their XML documents. Available together with these standard DTDs could be the software modules that process them.

Example: Combining several DTDs into one document

```
<?XML version = "1.0" ?>
<DOCTYPE Document[
<!ENTITY     header       SYSTEM header.dtd>
<!ENTITY     body         SYSTEM body.dtd>
<!ENTITY     footer       SYSTEM footer.dtd>
<!ELEMENT    Document     (%header, %body, %footer)>
]>
```

1.3.4 XML and Datatypes

One of the first reactions a programmer has when confronted with a new language is to find out what datatypes it supports. With datatypes I mean integers, characters, strings, lists, and so on. The beauty of XML is that it supports all datatypes and none. It is up to the application parsing the XML source file to determine what datatype that a certain element has. XML will not make any type checking of the data stored. The only checks that will be performed are against the structure (valid and well-structured) of the document.

However there is a need to be able to describe datatypes in XML documents in a standardized and consistent way. Therefore an extension specification to XML has been proposed called XML-Data. This specification has been accepted as a W3C Note and is described later.

More complex datatypes

Since XML treats all elements the same it is very easy to define and implement more complex datatypes as well. Here is an example of an XML document describing a binary tree

Example: A binary tree in XML

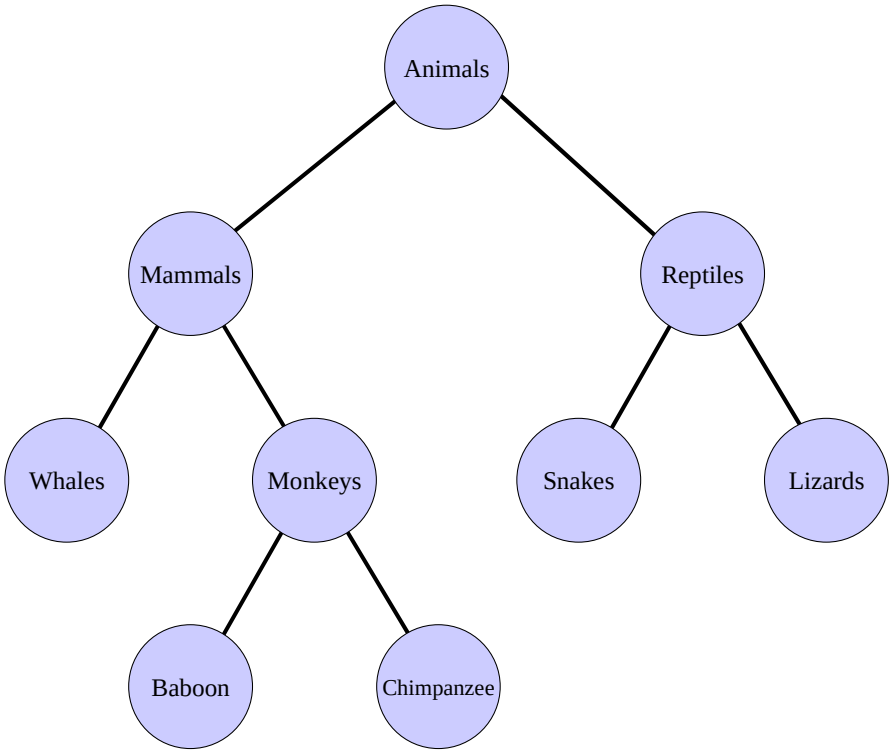


Figure 2: Shows a graphical representation of the binary tree (containing animals).

```
<?XML version = “1.0” ?>
<!DOCTYPE BINARY_TREE [
<!ELEMENT BINARY_TREE (ROOT)>
<!ELEMENT ROOT (VALUE,CHILDREN)>
<!ELEMENT CHILDREN(LEFT_CHILD?,RIGHT_CHILD?)>
<!ELEMENT LEFT_CHILD(VALUE?, CHILDREN?)>
<!ELEMENT RIGHT_CHILD(VALUE?, CHILDREN?)>
<!ELEMENT VALUE(#PCDATA)>]>

 <BINARY_TREE>
       <ROOT>
             <VALUE>Animals</VALUE>
             <CHILDREN>
                   <LEFT_CHILD>
                         <VALUE>Mammals</VALUE>
                         <CHILDREN>
                               <LEFT_CHILD>
                                     <VALUE>Whales</VALUE>
                               </LEFT_CHILD>
                               <RIGHT_CHILD>
                                     <VALUE>Monkeys</VALUE>
                                     <CHILDREN>
                                           <LEFT_CHILD>
                                                 <VALUE>
                                                       Baboon
                                                 </VALUE>
                                           </LEFT_CHILD>
                                           <RIGHT_CHILD>
                                                 <VALUE>
                                                       Chimpanzee
                                                 </VALUE>
                                           </RIGHT_CHILD>
```

```
                                        </CHILDREN>
                                    </RIGHT_CHILD>
                                </CHILDREN>
                            </LEFT_CHILD>
                            <RIGHT_CHILD>
                                <VALUE>Reptiles</VALUE>
                                <CHILDREN>
                                    <LEFT_CHILD>
                                        <VALUE>Snakes</VALUE>
                                    </LEFT_CHILD>
                                    <RIGHT_CHILD>
                                        <VALUE>Lizards</VALUE>
                                    </RIGHT_CHILD>
                                </CHILDREN>
                            </RIGHT_CHILD>
                        </CHILDREN>
                    </ROOT>
</BINARY_TREE>
```

This concludes my very brief introduction to XML syntax. If you want to read more about the XML syntax there are many good books on the subject, I can particularly recommend *XML Complete* by Steven Holzner [16].

1.4 XML and Java – friends or enemies?

XML in many ways augments Java; however, XML is also evolving into an object transport protocol that could undermine Java's claim as a does-all platform. XML tags Web-based information for recognition by developers and computers, which is necessary because HTML lacks a way to add meaning to content aside from cryptic URLs. XML aims to add that meaning to Web objects, a task once assigned to Java.

"XML lets developers choose between building Web applications or Java systems."
-- Adam Berrey, product marketing director at Allaire, a maker of Web application servers.

All that now is needed to make advanced Web applications is a client that renders XML information directly into the client-side browser. No longer are Java virtual machine (JVM)-based clients the only choice.

"Ultimately, XML will obviate the need for a JVM on the client."
-- Doug Pollack, vice president of marketing at GemStone Systems, in Beaverton, Ore.

Sun Microsystems, the company behind Java, acknowledges that XML can communicate to clients without JVMs, but it claims XML needs Java to reach its full potential.

"XML is not useful alone; it is a complement to Java. Java provides the portable code to XML, and XML offers the data for Java."
-- Nancy Lee, product manager for XML at Sun Microsystems.

XML can create open data that is not dependent on a platform, language, or restrictive formatting convention. If widely adopted, XML could become a de facto standard for communicating content and objects down to clients. That sounds a lot like what Java does, at least as a content platform. It is just such a role for XML that appeals to Microsoft, which disdains Java's use for purposes other than programming.

"XML lets you exchange information across platforms, not to be confused with writing cross-platform applications."
-- Dave Wascha, XML product manager at Microsoft.

But for many, it is not a matter of choosing between Java and XML. Many Web developers have come to the conclusion that XML and Java is the perfect pair because they complement each other so well. XML contributes platform-independent *data* - portable documents and data. Java contributes platform-independent *processing* - portable object oriented software solutions.

The applications that will drive the acceptance of XML are those that cannot be accomplished within the limitations of HTML. These applications can be divided into four broad categories:

- Applications that require the Web client to mediate between two or more heterogeneous databases.
- Applications that attempt to distribute a significant proportion of the processing load from the Web server to the Web client.
- Applications that require the Web client to present different views of the same data to different users.
- Applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users.

"We're doing XML and Java, and see them as wonderfully complementary."
-- David Skok, chairman and founder of SilverStream Software.

"A Java strategy without an XML strategy is incomplete."
-- Eric Brown, an analyst at Forrester Research, in Cambridge.

This corresponds well with what IBM, a driving force behind both Java and XML, seems to have in mind.

"XML and Java are parallel and symbiotic. Both are crucial to computing in the new millennium"
-- Simon Phipps, head of XML marketing and Java evangelist at IBM.

Part 2: Related Specifications

2.1 W3C Specifications and Levels

The specifications described in this paper are still under development by the W3C and are more or less subject to change. Depending on how far a specification has progressed, it is placed different W3C specification levels. I will here give a listing of the different levels and which specifications are currently on what level. All specifications listed here will be described later.

There exist four different levels of the W3C specifications:

- **Recommendations** - signifying that the specifications are stable, contribute to Web interoperability, and are supported for industry-wide adoption by the W3C Membership.
  - Extensible Markup Language (XML) 1.0 specification [1].
  - Document Object Model (DOM) Level 1 specification [8].
  - Cascading Style Sheets Level 1 (CSS1) specification [2].
  - Cascading Style Sheets Level 2 (CSS2) specification [3].
  - Namespaces in XML (XML Namespace) specification [5].

- **Proposed Recommendations** - signifying that the specifications are under review by W3C Members.
  - Currently there are no XML related proposed recommendations.

- **Working Drafts** - signifying that the specifications may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C working drafts as reference material or to cite them as other than "work in progress".
  - Extensible Stylesheet Language (XSL) specification [4].
  - XML Linking Language (XLink) specification [6].
  - XML Pointer Language (XPointer) specification [7].

- **Notes** - the W3C may make available on the Web information, ideas or commentary from W3C staff, Members, or the general public. Such information may be released, at the discretion of the W3C Director, as a Note.
  - XML Query Language (XML-QL) specification [10].
  - XML Data (XML-Data) specification [52].
  - Schema for Object-oriented XML (SOX) specification [9].
  - Vector Markup Language (VML) specification [12].

2.2 Document Object Model (DOM)

The W3C Document Object Model (DOM) [8] is a platform- and language-neutral program interface that will allow programs and scripts to access and update the content, structure, and style of documents in a standard way. This standard interface will make reliable scripting across platforms a reality. The DOM is designed with both HTML and XML in mind. Interoperability is a necessity for documents sent over the Internet, but this is not the only benefit of having a standard Applications Programming Interface (API). A standard API will also make it easier to develop modules that can be re-used in different applications.

"XML on its own is really this stream of internationalized characters that follows certain rules. But you can't actually do anything with this stream of characters with pointy brackets and question marks in it. But, for example, if you have used XML to define items in a catalog, you need some way of getting ordering information out of the catalog and inside an order form, inside that commerce application. That's the DOM."
--Lauren Wood, chairperson of the W3C DOM working group

The Document Object Model is, despite its name, not an object model in the same way as the Component Object Model (COM) [20,37,40]. The COM, like CORBA [37,39], is a language-independent way to specify interfaces and objects; the DOM is a set of interfaces and objects designed for managing HTML and XML documents.

There will be three ways of using the DOM to access XML documents:

- Using JavaScript or VBScript in the Web page.

- Using an external application such as a plug-in, or ActiveX control, that accesses the document through the browser.
- Using an external XML parser that implements the DOM.

Both Netscape Navigator and Microsoft Internet Explorer have their own proprietary DOMs, but both companies say they will support the W3C standard DOM in the next versions of their browsers.

The W3C DOM consists of different levels, each new level is based on the previous and extends the API to make it more usable. These levels together are usually referred to as the DOM Specification.

- **Level zero**: Functionality equivalent to that evident in Netscape Navigator 3.0 and Microsoft Internet Explorer 3.0. The W3C DOM builds on this existing technology.
- **Level one**: This level concentrates on the actual core, HTML, and XML document models. It contains functionality for document navigation and manipulation. It has recently been approved as a W3C Recommendation [8].
- **Level two:** will include a style sheet object model, and define functionality for manipulating the style information attached to a document. It will also allow rich queries of the document and define some event model.
- **Further levels**: These will specify some interface to the possibly underlying window system, including some ways to prompt the user. They will also contain functions to manipulate the document's DTD. Finally, they will include some security model.

The DOM, however, ***"isn't a silver bullet. Even with it, you won't necessarily be able to write full-featured, fully interoperable applications. That's a misconception"***
-- Sara Williams, lead product manager for Microsoft's Internet Explorer browser software.

For one thing, the DOM provides access only to the elements that make up HTML and XML documents, it does not say how those elements are manipulated. So, for instance, if a JavaScript is used, its syntax must be equally supported across both major browsers. Complicating the situation, DOM Level 1 does not define an event model, leaving that matter to the individual browsers.

2.3 Stylesheet Languages

As has been previously stated, one of the most important aspects of XML is that it separates style from data. This is great because it enables different users to define their own views of how they want the data to be presented. So if there is no information about presentation in the XML documents, then how is this information stored?

The answer is that the information is stored separately in a stylesheet document. Stylesheets offer precise control over the presentation of Web pages. A stylesheet is a set of stylistic rules that describe how Web documents are presented to users. Using stylesheets you can specify such things as for instance, the size, color, and spacing of text. It can also specify the placement of text and images on the page, plus a whole lot more.

W3C continues to work with its members, evolving two such stylesheets languages to use with XML; the Cascading Style Sheets (CSS) language, and the Extensible Stylesheet Language (XSL). Both are described in more detail below in their own chapters.

The fact that W3C has started developing XSL in addition to CSS has caused some confusion. Why develop a second style sheet language when implementers haven't even finished the first one? The answer can be found in the table below:

	CSS	**XSL**
Can be used with HTML?	yes	no
Can be used with XML?	yes	yes
Transformation language?	no	yes
Syntax	CSS	XML

The unique features are that CSS can be used to style HTML documents. XSL, on the other hand, is able to transform documents. For example, XSL can be used to transform XML data into HTML/CSS documents on the Web server. This way, the two languages complement each other and can be used together. Both languages can be used to style XML documents.

CSS and XSL will use the same underlying formatting model and designers will therefore have access to the same formatting features in both languages. W3C will work hard to ensure that interoperable implementations of the formatting model is available.

2.4 Cascading Style Sheets (CSS)

CSS is a simple declarative language that allows authors and users to apply stylistic information (concerning font, spacing, color, and so on) to structured documents written in HTML or XML. Designers specify how elements are rendered by associating them with *properties* and *values*.

For example -

```
H1       {
          font-size: 12pt;
          font-weight: bold;
          color: blue;
         }
```

- declares that H1 elements should be 12 point bold blue text. The many other CSS properties allow you to specify everything from the font and color of individual paragraphs, headings and other text, to the size of margins, the distance between lines, the type of bullet, "white space" around images, background textures and a great deal more.

As is the case with the W3C DOM, the CSS specification also consists of different levels.

1. **CSS Level 1 (CSS1)** [2] - has been a W3C Recommendation for more than two years, it was approved in December 1996. CSS1 is a simple stylesheet mechanism that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML documents. The CSS1 language is human readable and writable, and expresses style in common desktop publishing terminology. As the name implies more than one style sheet can "cascade" together to produce the final look of the document; individual style sheets from different sources can be combined.
2. **CSS Level 2 (CSS2)** [3] – became a W3C Recommendation in May 1998. CSS2 builds on CSS1 and, with very few exceptions, all valid CSS1 style sheets are valid CSS2 style sheets. CSS2 is described in more detail in the following chapter.

2.4.1 Cascading Style Sheets Level 2 (CSS2)

CSS2 is a style sheet language that allows authors and users to attach style (e.g., fonts, spacing, and aural cues) to structured documents (e.g., HTML documents and XML applications). CSS2 includes all the power of CSS1, and adds enhancements in several areas to make the Web more appealing for both content providers and users. Although originally developed for HTML, CSS has been designed to allow you to style XML documents also.

CSS2 has a number of new features including the following:

- **CSS now caters for "paged media"** - which mainly relates to paper or transparencies. CSS2 uses a page model, which specifies how a document is formatted within a rectangular area called the page box to control how, the document will look when printed.
- **Style sheets can point to fonts on the Web**: in CSS2 you can specify the desired font characteristics in great detail. A new feature allows designers to specify where to download a part of the font needed to display a document. This will no doubt become a popular approach.
- **CSS gives you the ability to specify rectangular regions on a page**. CSS2 provides ways for defining rectangular regions for displaying different parts of documents, and these can overlap and show through as required. This idea for positioning elements gives authors the freedom to layout documents as they want, without using the HTML table element.
- **CSS2 allows you to invent style sheets for different "media types"**. For display on a color computer screen, for example, a style sheet might concentrate on color and layout. The style sheet for rendering the document into speech, would instead focus on the pitch of the voice, the volume, and so on. CSS media types cover speech synthesizers, braille printers, small handheld devices, slide projects, to name just a few.

2.4.2 Viewing XML using CSS2 (An Example)

CSS can be used with any structured document format; we are interested in using it together with XML. Here is an example of a simple XML fragment:

```
<ARTICLE>
        <HEADLINE>Fredrick the Great meets Bach</HEADLINE>
        <AUTHOR>Johann Nikolaus Forkel</AUTHOR>
        <PARA>
                One evening, just as he was getting his
                <INSTRUMENT>flute</INSTRUMENT> ready and his
                musicians were assembled, an officer brought him a list of
                the strangers who had arrived.
        </PARA>
</ARTICLE>
```

To display this XML fragment in a document-like fashion using CSS2, we have to do two things:
(i) create an appropriate stylesheet, and
(ii) associate the XML fragment with that stylesheet.

2.4.2.1 Creating the CSS2 stylesheet

The stylesheet is simply a text file with the extension `.css`. For this example I create a text file called `bach.css`, in this text file I put my styling instructions.

First I must declare which elements are inline-level (i.e. do not cause line breaks) and which are block-level (i.e. cause line breaks).

```
INSTRUMENT {display: inline}
ARTICLE, HEADLINE, AUTHOR, PARA {display: block}
```

The first rule declares `INSTRUMENT` to be inline and the second rule, with its comma-separated list of selectors, declares all the other elements to be block-level.

Thereafter to get the wanted look I insert my styling instructions. For example, the headline font size should be larger than then rest of the text, and I want to display the author's name in italic:

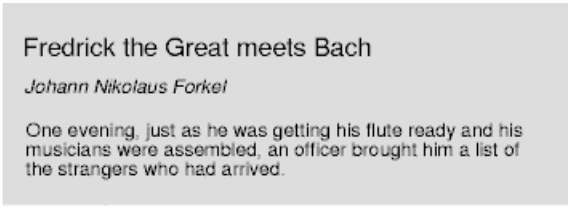
```
INSTRUMENT {display: inline}
ARTICLE, HEADLINE, AUTHOR, PARA {display: block}
HEADLINE {font-size: 1.3em}
AUTHOR {font-style: italic}
ARTICLE, HEADLINE, AUTHOR, PARA {margin: 0.5em}
```

2.4.2.2 Associating the CSS stylesheet

One proposal for linking a style sheet to an XML document is to use a processing instruction inline in the document itself :

```
<?XML:stylesheet type="text/css" href="bach.css"?>
```

Now, using the two together will produce something like the following result:



For more information about CSS and especially CSS2 I refer to the CSS1 and CSS2 specifications [2,3].

2.5 Extensible Stylesheet Language (XSL)

CSS is a good choice for documents where elements - paragraphs, lists, headings, tables, and so on, are typically rendered in the same order as they are specified in the source document. Sometimes, however, you may want to re-order elements. For example, you might want to generate a table of contents by selecting the text of headings, or generate a report document based on information currently represented in XML. These sorts of task require a different kind of operation; XSL allows you to do these sorts of things. Written in XML, the syntax for XSL is still under development and subject to change.

2.5.1 Origin

XSL stems from two standards, Document Style Semantics and Specification Language (DSSSL) and the previously described Cascading Style Sheets (CSS) language.

DSSSL is an ISO standard [17] for specifying document transformation and formatting in a platform- and vendor-neutral manner. DSSSL is used to specify the presentation of documents marked up using the Standard Generalized Markup Language (SGML) [43]. DSSSL consists of two main components: a transformation language and a style language. The transformation language is used to specify structural transformations on SGML source files. For example, a telephone directory structured as a series of entries ordered by last name could, by applying a transformation specification, be rendered as a series of entries sorted by first name instead. The transformation language can also be used to specify the merging of two or more documents, the generation of indexes and tables of contents, and other operations. The style language provides a standardized, powerful language for describing the formatting of SGML documents.

"It's a mistake to put DSSSL into the same bag as scripting languages. The DSSSL stylesheet is one giant function whose value is an abstract, device-independent, nonprocedural description of the formatted document that gets fed as a specification of display areas to downstream rendering processes" -- Jon Bosak, Chair of the W3C XML Group.

XSL combines these two standards, DSSSL and CSS, and also incorporates the full power of a programming language to provide advanced style functionality and interactivity. XSL embeds the language ECMAScript [53], a standardized version of JavaScript, to provide this functionality.

As with XML, W3C has put up some design goals that help us understand the usage of XSL.

1. XSL should support browsing, printing, interactive editing and design tools.
2. XSL should be capable of specifying presentations for traditional and Web environments.
3. XSL should support interaction with structured information, as well as presentation of it.
4. XSL should support all kinds of structured information, including both data and documents.
5. XSL should support both visual and non-visual presentations.
6. XSL should be a declarative language.
7. XSL should be optimized to provide simple specifications for common formatting tasks and not preclude more sophisticated formatting tasks.
8. XSL should provide an extensibility mechanism.
9. The number of optional features in XSL should be kept to a minimum.
10. XSL should provide the formatting functionality of *at least* DSSSL and CSS.
11. XSL should leverage other recommendations and standards, including XML, XLL, DOM, HTML and ECMAScript.
12. XSL should be expressed in XML syntax.
13. XSL stylesheets should be human-readable and reasonably clear.
14. Terseness in XSL markup is of minimal importance.

2.5.2 How does XSL work?

XSL is a language for expressing stylesheets. To construct a view of the stored data, a XSL processor uses the XSL stylesheet to parse an XML source document and construct usable output. At the moment the only XSL processors available provides output in HTML, but in theory this output could be anything: sound-files, RTF, raw text, etc. The XSL standard sets no limitations on the output. In the future XSL will hopefully be supported in the Web browsers directly, but currently there is no such support.

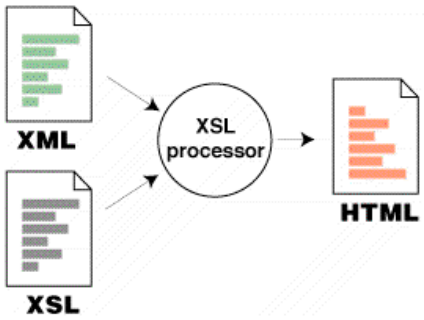


Figure 3: Shows how an XSL processor uses the XSL stylesheet to parse an XML source document to construct HTML output datatypes.

So, how does the XSL processor know which XSL file that contains the stylesheet for a certain XML document? The answer is that the XSL file in question is named in the XML document, using a standardized stylesheet-tag (as was also the case with CSS):
`<?xml-stylesheet type="text/xsl" href="www.book.com/s1.xsl">`.

Each XSL stylesheet contains a set of template rules for presenting a class of XML source documents. These template rules have two parts, one *pattern* part that identifies the elements in the XML source document that this rule applies to, and one *action* part that defines what should be done with the element. These patterns and actions are then used to map the source tree (remember that the XML source code defined a logical tree) to a result tree consisting of flow objects that define the user interface.

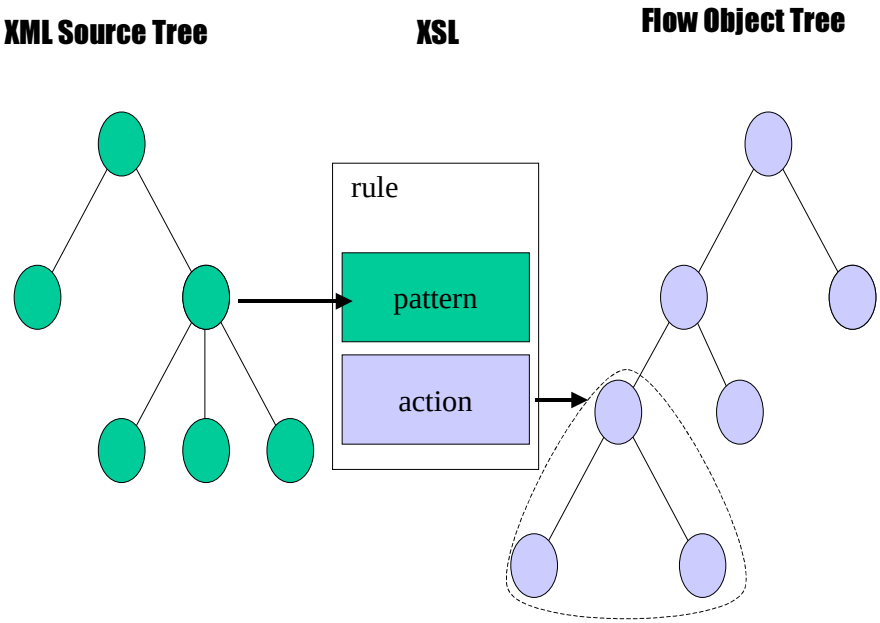


Figure 4: Shows how a template rule maps between the XML source tree and a flow object tree.

2.5.2.1 Patterns

The *patterns* are used to identify which rules are applicable for a given source element. A pattern is a string, which is matched against all elements in the XML source document. The most common pattern specifies the element tag name of a matching element. For example, the pattern `Car` matches an element whose tag name is `Car`. More complex patterns specify the element tags of ancestors of a matching element. For example, the pattern `Vehicle/Car` matches an element whose tag name is `Car` that has a parent element with tag name `Vehicle`. The patterns can be made very complex to single out a specific element.

2.5.2.2 Actions

When a pattern identifies (or matches) a type of XML source element, the *action* portion of the rule is used to create the formatted output. The action describes what the output structure should be, what formatting properties to apply, and how to process the children of the source element. The formatted output is also known as *flow objects.*

2.5.2.3 Flow Objects

The flow objects are what actually make up the visual output. Think of each item on the user interface as a flow object. Flow objects can be text paragraphs, images, tables, lists, and so on. Conceptually these objects form a tree. Paragraphs, tables, sequences and other “container” objects are the branches, and characters, images and other “atomic” objects are the leaves. The output document is the root of the tree. The tree of flow objects is simply called the flow object tree.

Every flow object has characteristics. The exact set of characteristics that a flow object has depends on its class. For example, Web pages have scrollbars, clickable links have destinations, fonts have font sizes and pictures have heights and widths.

XSL have a special set of flow objects that are called “HTML flow objects” and they correspond to the element types in the HTML DTD. If they are used to format XML documents it will appear to the user as if they have been created using HTML directly. This is useful since there are not many browsers available today that support XSL but HTML is a common standard. Another such set of flow objects are called “DSSSL flow objects”, they correspond to the DSSSL element types. Currently there is no tool available that can handle DSSSL flow objects.

Microsoft – XSL and CSS

In January 1999, the U.S. Patent Office awarded Microsoft a patent that could have a major impact on Web standards. The U.S. Patent No. 5,860,073, which broadly covers "the use of style sheets in an electronic publishing system," appears to describe some of the key concepts used in the W3C’s Cascading Style Sheets (CSS) and Extensible Stylesheet Language (XSL) standards. The patent could potentially require these currently open standards to be licensed from Microsoft.

Microsoft claims that this patent could actually protect Web standards by preventing other vendors from engaging in "standards terrorism" with intellectual property claims of their own. That comment strikes me as hilarious. While it can't be proven that Microsoft deliberately filed the patent in order to get a proprietary grip on the standards, the fact that Microsoft didn't reveal the filing during the CSS definition process shows bad faith toward the W3C and its process.

The Web Standards Project (WSP) [http://www.webstandards.org], an international coalition of Web developers, has reacted very strongly to the patent and has called on Microsoft and the W3C to clarify whether a Microsoft patent gives the company control over two key Web standards (CSS and XSL) developed by W3C.

If the CSS and XSL standards are in fact covered by the patent, WSP believes Microsoft, which participated in W3C's development of these standards, should immediately take legal steps to ensure these Web standards remain openly available on a nondiscriminatory basis. This could include turning over the patent to the W3C, or other legal licensing agreements that irrevocably protect these open standards. WSP also called on any other companies that may be pursuing other patents that affect W3C standards to take similar measures.

WSP also questioned whether the U.S. Patent Office should have granted Microsoft’s patent on “style sheets”. WSP’s reasons are that there already exist a number of prior examples of similar technology, including the original proposal for CSS. Microsoft's patent claims its innovation is to apply style sheets to text on-the-fly when the document is displayed on a user's computer. However, that same technology has been used on several different batch pagination systems, dating back to the 1960s, which have been used for book, directory, and database publishing.

"We’d be opposed to any private company holding control over an open standard. There are inherent conflicts of interest there when you’re asking a company to license this open standard to potential competitors. The best way we see to resolve that situation is to hand over that license to the W3C”
-- George Olsen, project leader for the Web Standards Project (WSP).

2.6 Extensible Linking Language (XLL)

One of the most popular features of HTML today is the ability to insert links in a Web page to other related Web pages and thus creating the World Wide Web. XML fully incorporates this idea and extends it beyond its current limitations. XLL is a broad term for XML hyperlinking (linking and addressing) and it has two major components: XLink and XPointer.

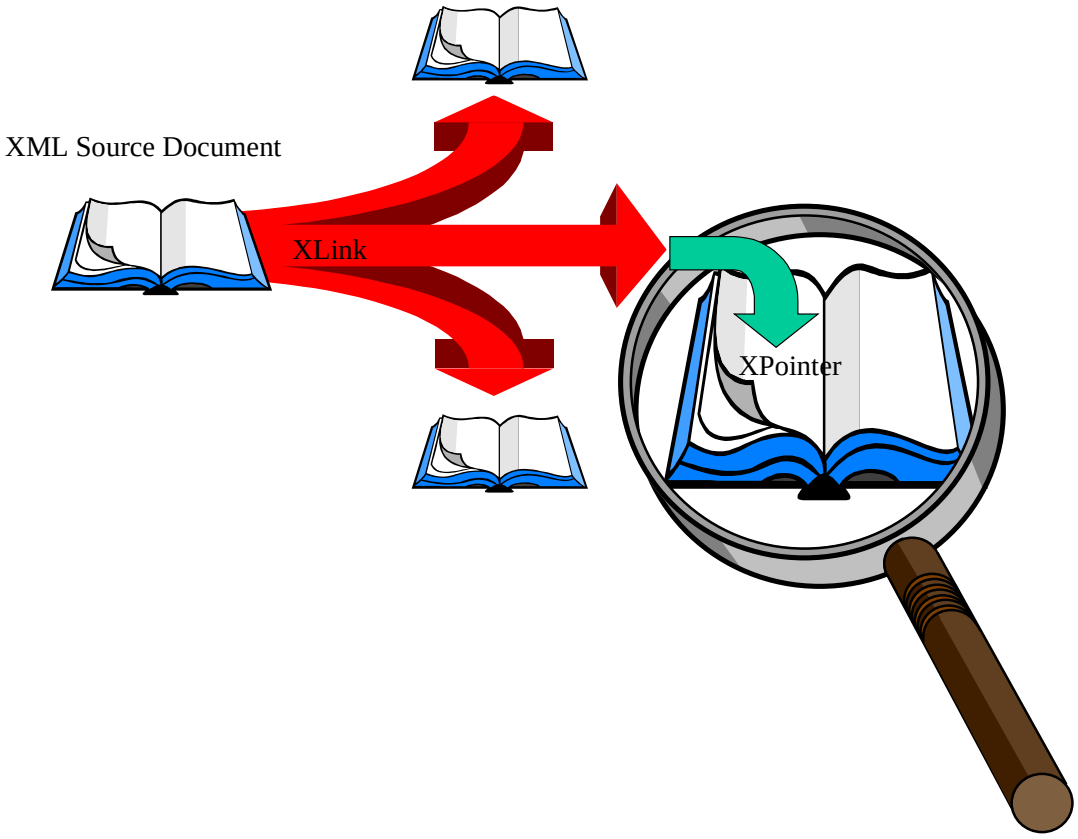


Figure 5: Shows how the XLink and XPointer work together in XLL. The XLink specification locates the resources and the XPointer specification points within that resource.

2.6.1 Origin

Three standards have been especially influential when constructing XLL:

- **HTML**: Defines several SGML element types that represent links.
- **HyTime** [44]: Defines inline and out-of-line link structures and some semantic features including traversal control and presentation of objects.
- **Text Encoding Initiative Guidelines** [45]: Provide structures for creating links, aggregate objects, and link collections.

2.6.2 XLink

The XLink draft [6] specifies constructs that may be inserted into XML source documents to describe links between objects. A link, as the term is used here, is an explicit relationship between two or more data objects. XLink uses XML syntax to create structures that can describe the simple unidirectional hyperlinks of today's HTML as well as more sophisticated multi-ended and typed links.

Following is a summary of the design principles governing XLink:

1. XLink shall be straightforwardly usable over the Internet.
2. XLink shall be usable by a wide variety of link usage domains and of classes of linking application software.
3. The XLink expression language shall be XML.
4. The XLink design shall be prepared quickly.

5. The XLink design shall be formal and concise.
6. XLinks shall be human-readable.
7. XLinks may reside outside the documents in which the participating resources reside.
8. XLink shall represent the abstract structure and significance of links.
9. XLink must be feasible to implement.

2.6.2.1 How does XLink work?

XLink's unidirectional hyperlinks work precisely like HTML links work today:

- The link is expressed at one of its ends (similar to the `<A HREF = A HTTP reference>` element in HTML documents) .
- Users can only initiate travel from that end to the other.
- The links effect on windows, frames, go-back lists, stylesheets in use, and so on, is mainly determined by browsers, not by the link itself. For example, traversal of `A` links normally replaces the current view, perhaps with a user option to open a new window.
- The link goes to only one destination (although a server may have great freedom in finding or dynamically creating that destination).

While this set of characteristics already is very powerful and obviously has proven itself highly useful and effective, each of these assumptions also limits the range of hypertext functionality. XLink provides ways to create links that go beyond each of these specific characteristics, thus providing features previously available mostly in dedicated hypermedia systems.

- **Multi-directional links** - HTML only provides for one-way links. XLink provides multi-directional links (note that "go back" is not at all the same thing; a multi-directional links can be traversed in either direction *regardless of whether you went the other way first).*
- **Links with multiple destinations** - Users may be able to choose between different destinations from a single link.
- **Links with types** - Users can define links to be of a certain type. The link-types can then be used to identify links that should be processed similarly.
- **Databases for organizing link locations** - Currently HTML links rely upon fixed machine and file system addresses to find information. XLink provides the framework for link databases to store these addresses. When kept up-to-date, link databases will free HTML publishers from maintaining frequently changing link locations.
- **Links that annotate read-only documents** - That is, XLink provides the possibility for users to add links to a page even though they don't own that page. Of course, this involves a process of deciding whose links you want to display; but this also makes it possible to build a valuable infrastructure of annotation, commentary, and communal evaluation and discussion on the Web.

2.6.3 XPointer

The XPointer draft [7] specifies constructs that support addressing into the internal structures of XML documents. In particular, it provides for specific reference to elements, character strings, and other parts of XML documents, whether or not they bear an explicit ID attribute. This differs from the HTML links that can only point to elements in a document if they have been anchored.

Following is a summary of the design principles governing XPointer:

1. XPointers are addressed into XML documents.
2. XPointers shall be straightforwardly usable over the Internet.
3. XPointers shall be straightforwardly usable in URIs.
4. The XPointer design shall be prepared quickly.
5. The XPointer design shall be formal and concise.
6. The XPointer syntax shall be reasonably compact and human readable.
7. XPointers shall be optimized for usability.
8. XPointers must be feasible to implement.

XPointer provides better locations specifications than HTML:

- **Pointers without anchors** - Links that point to specific places *inside* of documents, even when the author of those documents didn't already provide an ID at just the right place.
- **Better addressing** - Fine-grained addressing to elements, character strings, and spans inside documents.

- **Clear syntax** - A clear syntax for talking about locations and relationships in hierarchies (such as the structure of XML documents), so that locations are human-readable and writable, rather than mere hash.

2.7 XML – Namespaces

One of the advantages with XML is the ability for the programmer to create their own tag labels and elements for the data they want to transfer. One potential problem with this is that name collisions can occur when many XML documents are going to be combined into one. To avoid this problem W3C has proposed the usage of namespaces within XML.

[Definition:] An **XML namespace** [5] is a collection of names, identified by a URI, which are used in XML documents as element types and attribute names.

"One of the interesting things you can do with XML is get information from multiple data sources and put it in one document, Namespaces lets you know which information came from which site."
-- Dave Wascha, XML product manager at Microsoft.

Quite a few people, after reading earlier drafts of the Namespace Recommendation, decided that namespaces were actually a facility for modular DTDs, or were trying to duplicate the function of SGML's "Architectural Forms". None of these theories are true. The only reason namespaces exist is to give elements and attributes programmer-friendly names that will be unique across the whole Internet.

Namespaces are a simple and straightforward way to distinguish names used in XML documents, no matter where they come from. However, the concepts are a bit abstract, and this specification has been causing some mental indigestion among those who read it.

An XML document contains a tree of elements. Each element has an element type name (sometimes called the tag name) and a set of attributes; each attribute consists of a name and a value. Applications typically make use of the element type name and attributes of an element in determining how to process the element. In XML 1.0 without namespaces, element type names and attribute names are unstructured strings using a restricted set of characters, similar to identifiers in programming languages. I will call these names *local names*. This is problematic in a distributed environment like the Web. One XML document may use `<part>` elements to describe parts of books, another may use `<part>` elements to describe parts of cars. An XML application has no way of knowing how to process a `<part>` element unless it has some additional information external to the document.

The XML Namespaces Recommendation tries to improve this situation by extending the data model to allow element type names and attribute names to be qualified with a URI. Thus a document that describes parts of cars can use `<part>` qualified by one URI; and a document that describes parts of books can use `<part>` qualified by another URI. I will call the combination of a local name and a qualifying URI a *universal name*. The role of the URI in a universal name is purely to allow applications to recognize the name. There are no guarantees about the resource identified by the URI.

2.7.1 Using Namespaces – XML Syntax

A namespace is **declared** using an attribute whose prefix is `xmlns`. The namespace is located by a URI which functions as a location combined with a global **namespace name** to identify the namespace. The namespace name, to serve its intended purpose, should have the characteristics of uniqueness and persistence. The namespace can be given a local name. If no local name is given for the namespace it is assumed that the namespace is the **default namespace** in the scope of the element to which the declaration is attached.

The best way to understand namespaces, as with many other things on the Web, is by example. One important detail to look at and understand is the scope of the namespace in the different kinds of declarations.

Example 1:

```
<x xmlns:edi='http://ecommerce.org/schema'>
<!-- the "edi" prefix is bound to http://ecommerce.org/schema for the "x"
```

```
element and contents -->
</x>
```

This example shows how namespaces are declared and what scope they get as standard.
`<x xmlns:edi="http//ecommerce.org/schema">`

- `x` is the markup tag name for this element.
- `xmlns` (abbreviation for XML NameSpace) is an attribute name signifying that a namespace is used inside of the element `x` if nothing else is said later.
- The HTTP-address locates where on the Internet the specific `schema` namespace is defined. The local name for that namespace is set to `edi`.

Example 2:

```
<x xmlns:edi='http://ecommerce.org/schema'>
<!-- the 'price' element's namespace is http://ecommerce.org/schema -->
<edi:price units='Euro'>32.18</edi:price>
</x>
```

Shows how a namespace is used in element declarations. Using the namespace defined in the previous example uniquely identifies the `price` element. We also see the usage of an attribute named `units` within the `price` tag.

Example 3:

```
<?xml version="1.0"?>
<!-- all elements here are explicitly in the HTML namespace -->
<html:html xmlns:html='http://www.w3.org/TR/REC-html40'>
	<html:head>
		<html:title>Frobnostication</html:title>
	</html:head>
	<html:body>
		<html:p>Moved to
			<html:a href='http://frob.com'>here.</html:a>
		</html:p>
	</html:body>
</html:html>
```

This more complex example shows how to specify in which namespace a certain tag name is defined.
`<html:html xmlns:html-def="http//www.w3.org/TR/REC-html40">`

- `html:html` sets the tag name for this element to `html` and at the same time says that this tag name is predefined in the local namespace `html`.
- Remaining definitions are the same as in the above example.

Example 4:

```
<?xml version="1.0"?>
<!-- initially, the default namespace is "books" -->
<book xmlns='urn:loc.gov:books' xmlns:isbn='urn:ISBN:0-395-36341-6'>
	<title>Cheaper by the Dozen</title>
	<isbn:number>1568491379</isbn:number>
	<notes>
		<!-- make HTML the default namespace for some commentary -->
		<p xmlns='urn:w3-org-ns:HTML'>
			This is a <i>funny</i> book!
		</p>
	</notes>
</book>
```

This example shows the scope of different types of definitions.
`book xmlns='urn:loc.gov:books'`

- Defines the standard namespace within the element `book` to be `urn:loc.gov:books`. All the tag labels within `book` will automatically be assumed to be inside this namespace if nothing else is said further down in the hierarchy.

`xmlns:isbn='urn:ISBN:0-395-36341-6'`

- Defines and names `isbn` as the local name for an additional namespace `ISBN` that can be used within the element `book`. It is later used in `<isbn:number>1568491379</isbn:number>`.

`<p xmlns='urn:w3-org-ns:HTML'>`

- Defines the standard namespace within the element `p`.

These examples show the usefulness of the XML Namespace Recommendation, it is an important extension of XML.

2.8 XML-Data

In the XML specification, the contents of the tagged elements are always interpreted as a string. This is not satisfactory for all purposes since many applications need to be able to specify rigid constraints on the data they handle. They need to know if a certain data is an integer, float or a string. Some applications also need to specify within what range a certain value is allowed to be. The typical example is databases, which have very stringent constraints on their field-values.

In recognition of these needs, Microsoft, ArborText, DataChannel, Inso, and the University of Edinburgh in a joint proposal to the W3C suggested the XML-Data specification [52]. It was accepted as a W3C Note. The XML-Data specification provides a standardized way to describe datatypes, ranges, default values, and other information concerning XML elements.

In the specification, the datatype of an element is defined using a standardized datatype-namespace and a specific datatype attribute. Together this construct is referred to as the `dt:dt` attribute (the first dt is for the datatype–namespace and the second dt names the datatype-attribute). The value of the `dt:dt` attribute is a URI giving the datatype of a specific element. The URI can be explicitly in URI format or can rely on the namespace facility for resolution.

Example: Using the `dt:dt` attribute

```
<?xml version="1.0"?>
<!ELEMENT Book xmlns:dt="urn:uuid:C2F41010-65B3-11D1-A29F-00AA00C14882/”
       (Author, Title, Edition, Price)>
<!ELEMENT Author (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Edition (#PCDATA)>
<!ELEMENT Price (#PCDATA)>
<!ATTLIST Price dt:dt “fixed.14.4” #FIXED>
<!—Sets the datatype of all Price Elements to be of type “fixed.14.4” -->

<BOOK>
  <AUTHOR>Bjarne Stroustrup</AUTHOR>
  <TITLE>The C++ Programming Language</TITLE>
  <EDITION>Second Edition</EDITION>
  <PRICE>19.95</PRICE> <!-- Will automatically be of datatype: ”fixed.14.4” -->
</BOOK>
```

The `dt:dt` attribute is used to specify that the value in the price tag is of datatype `“fixed.14.4”` which is predefined in the namespace.

2.8.1 Current Support

Since Microsoft is one of the co-authors of the standard, there is already support for XML-Data in the Internet Explorer 5.0 (IE5.0) beta DOM. This does not mean that it will be incorporated in the W3C DOM. Neither does it mean that W3C-Data will be approved as a W3C Recommendation.

Here is a listing of the datatypes that IE5.0 supports.

Data Type	Parse Type
string	Pcdata
number	A number, with no limit on digits, can potentially have a leading sign, fractional digits, and, optionally, an exponent. Punctuate using U.S. English rules.
int	A number, with no optional sign, no fractions, no exponent
float	Same as for "number"
fixed.14.4	Same as for "number," but no more than 14 digits to the left of the decimal point and no more than 4 to the right
boolean	"1" or "0"
dateTime.iso8601	A date in a subset ISO 8601 format, with optional time and no optional zone. Fractional seconds can be as precise as nanoseconds.
dateTime.iso8601tz	A date in a subset ISO 8601 format, with optional time and optional zone.

	Fractional seconds can be as precise as nanoseconds.
date.iso8601	A date in a subset ISO 8601 format (no time)
time.iso8601	A time in a subset ISO 8601 format, with no date and no time zone
time.iso8601tz	A time in a subset ISO 8601 format, with no date but an optional time zone
i1	A number, with optional sign, no fractions, no exponent
i2	A number, with optional sign, no fractions, no exponent
i4	A number, with optional sign, no fractions, no exponent
i8	A number, with optional sign, no fractions, no exponent
ui1	A number, unsigned, no fractions, no exponents
ui2	A number, unsigned, no fractions, no exponents
ui4	A number, unsigned, no fractions, no exponents
ui8	A number, unsigned, no fractions, no exponents
r4	Same as "number"
r8	Same as "number"
float.IEEE.754.32	Same as "number"
float.IEEE.754.64	Same as "number"
uuid	Hexadecimal digits representing octets, optional embedded hyphens that should be ignored
uri	Universal Resource Identifier
bin.hex	Hexadecimal digits representing octets
char	String
string.ansi	String containing only ASCII characters <= 0xFF

2.9 XML Query Language (XML-QL)

Given its flexibility, its likely that XML in the future will be used to exchange huge amounts of data on the Web, just as HTML now enables exchange of a vast number of documents. However, the availability of huge amounts of XML data poses several technical questions that the XML standard does not address. In particular:

- How will data be extracted from large XML documents?
- How will XML data be exchanged, e.g., by shipping XML documents or by shipping queries?
- How will XML data be exchanged between user communities using different but related DTDs?
- How will XML data from multiple XML sources be integrated?

Data extraction, transformation, and integration are all well-understood database problems. Their solutions often rely on a *query language*, either relational (SQL) or object-oriented (OQL). These query languages do not apply immediately to XML, because the XML data differs from traditional relational or object-oriented data. XML data, however, is very similar to a data model recently studied in the research community: the *semi-structured data model* [27].

Semi-structured data may include both structured and unstructured portions and is hence termed "semi-structured". Roughly speaking, semi-structured data is data that is neither raw data, nor very strictly typed as in conventional relational- or object-orientated database systems. The need for semi-structured data arises naturally in the context of data integration, even when the data sources are themselves well-structured.

The WWW provides numerous examples of semi-structured data. An HTML-page contains structuring primitives such as tags and anchors. But there is no or little restrictions about how these can be used. A typical example is a data source about restaurants in the Bay Area (from the Palo Alto Weekly newspaper) called Guide. It consists of an HTML file with one entry per restaurant and provides some information on prices, addresses, style, and reviews. Data in Guide resides in groups of text with some implicit structure. One can write a parser to extract the underlying structure. However, there is a large degree of irregularity in the structure since:

(i) restaurants are not all treated in a uniform manner (i.e., much less information is given for fast-food joints) and

(ii) information is entered as plain text by human beings that do not present the standard rigidity of your favorite data loader. Therefore, the parser will have to be tolerant and accept that it fails to parse portions of text, these will remain as plain text.

Generally, semi-structured data lacks or don't conform to an a-priori given schema, and is self-describing. Several research query languages [28,60,61,62] have been designed and implemented for semi-structured data.

To address the questions about data extraction and transformation from semi-structured XML documents. AT&T and Inria have sent a proposal to W3C for an XML Query Language (XML-QL) [10]. XML-QL can express *queries*, which extract pieces of data from XML documents, as well as *transformations*. These transformations can, for example, map XML data between different DTDs and integrate XML data from different sources.

The difference between the proposed XML-QL and XSL, which also is used to extract data from XML documents, is that XSL is intended primarily for specifying style and layout of XML documents. XML-QL supports more data-intensive operations, such as joins and aggregates, and has better support for constructing new XML data, which is required by transformations.

2.9.1 Using XML-QL

The XML-QL queries have been proposed to consist of two main constructs, a pattern and a construction rule. The pattern is used similarly to XSL's patterns to identify the elements that the rule applies to. The construction rule defines what should happen with the element.

Example: An XML-QL query

```
WHERE <book>
        <publisher><name>Addison-Wesley</></>
        <title> $t</>
        <author> $a</>
      </> IN "www.a.b.c/bib.xml"
CONSTRUCT <result>
            <author> $a</>
            <title> $t</>
          </>
```

The `WHERE`-part of the query is the pattern that the XML-QL processor tries to match to elements in the XML source document. The `CONSTRUCT`-part defines what the output XML-code will look like.

Since XML-QL hasn't even yet been approved as a W3C working draft it is at this time hard to determine if it will be used.

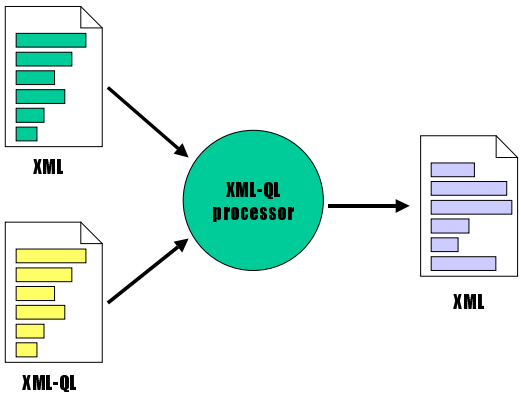


Figure 6: Shows how an XML-QL processor uses an XML-QL query to parse an XML source file and construct XML output.

2.10 Schema for Object-oriented XML (SOX)

In SGML the DTD defines for each SGML element, what possible other elements may be nested inside it. For example, in an invoice it may specify that the signing authority must be either Tom or Joe. It may specify that an item can be any part number or any accessory number or any book number. Checking the SGML validity of a document is a process that can be done automatically from the DTD. This is a check that do not verify semantic correctness, only structural correctness. But the structural constraints alone are

useful in many ways. For example, a user interface for constructing a document can be generated automatically from the structural constraints.

XML offers the Document Type Declaration (DTD) as formalism for defining the syntax and structure of XML documents. However, experience has shown that XML DTDs are not sufficient to specify content or semantics as in the above examples.

In recognition of this problem the Schema for Object-oriented XML (SOX) specification has been submitted to W3C as a Note [9]. SOX is used for defining the structure, content and semantics of XML documents to enable XML validation and higher levels of automated content checking. The SOX proposal has taken advantage of the XML 1.0 specification [1] as well as the XML-Data submission [52], the Document Content Description (DCD) submission [14] and the EXPRESS language reference manual [24].

SOX is proposed not only as an XML instance replacement syntax for SGML and XML document type definitions, but a modeling language for information modeling itself. SOX provides intrinsic datatypes, an extensible datatyping mechanism, content model and attribute interface inheritance, a powerful namespace mechanism, and embedded documentation.

SOX documents can be operated on by a SOX processor to produce many different types of output targets. Transformation of SOX documents will yield XML DTDs and object-oriented language classes to facilitate the development of intelligent applications, such as those needed to perform electronic commerce, for example. Other output targets of a schema include documentation derived from the documentation-based elements in SOX itself, and user interface components. Further output targets are yet to be defined, but the inherent flexibility of this schema language allows for many other options.

A SOX document, also known as a *schema*, is a valid XML document instance according to the SOX DTD, which represents a complete XML DTD-like structure. It has a document root element, and a representation of syntax that one would expect from a complete DTD, symbolically generated through the XML document instance.

2.10.1 Goals and Requirements

The goals of SOX are:

1. Schema language declaration constructs should be useful for the purpose of modeling markup relationships.
2. SOX documents, as compared with XML DTDs, should enable more efficient software development processes for distributed applications and dramatically decrease the complexity of supporting interoperation among heterogenous applications.
    - SOX should enable software mapping from SOX documents into data structures in relational databases, common programming languages, and interface definition languages (such as Java, COM, C and C++), resulting in usable code.
    - SOX should enable reuse at the document design and the application programming levels.
    - SOX should be able to express domain abstractions and common relationships among them directly and explicitly. (e.g., subtype/supertype, etc.)
    - SOX should support the generation of common application components (marshal/unmarshal, programming data structures) directly from SOX documents.

The requirements for SOX are:

1. SOX shall use XML syntax and be expressed in valid instances according to a valid XML DTD.
2. SOX and SOX documents shall be interoperable with XML software and conventions.
3. SOX shall enable a software mapping from SOX documents into an XML DTD, and from an XML DTD into a SOX document without losing the grammatical structure of the original DTD.
4. SOX shall provide an extensible datatyping mechanism.
5. SOX shall comply with and be compatible with applicable W3C recommendations, Internet Engineering Task Force (IETF) Request for Comments (RFCs) and ISO Standards, and Proposed Standards.
6. SOX documents shall provide support for embedded documentation.
7. SOX documents shall be human-readable.

2.10.2 Features

SOX is more expressive than the XML DTD in some of the following areas:

2.10.2.1 Base element types

SOX provide for parameterized base element types that can be used to build a foundation of regular patterns in SOX documents. You can define patterns such as tuples and triples, tabular and columnar data, business documents, indexes, bibliographies. Parameterization allows you to reuse the structure with different content model atoms in another document. Extending base element types allows you to add attributes or further specialize the attribute datatype, enumeration and presence. Code reuse on extended base element types is much higher than without.

2.10.2.2 Datatypes

SOX offer an extensive and extensible set of datatypes that may be applied to data content elements and attribute types. The purpose of datatypes is to provide a contract between parties as to the constraints that are applicable to data content in elements and attribute values. These constraints may be used by a content validation engine, prior to dispatch or upon receipt of an XML document or by user interface methods.

There are three varieties of datatypes in SOX documents: *scalar datatypes*, *enumerated datatypes*, and *format datatypes*. Scalar datatypes are derived from the basic *number* datatype, and support specification of the number of digits and decimal places, minimum and maximum value range, and a mask. An enumerated datatype may be derived from any of the intrinsic datatypes, and may specify an enumeration of valid values. A format datatype may be derived from any of the intrinsic datatypes, and must specify a mask.

SOX provide an extended list of intrinsic datatypes for attributes. Datatype extensibility is built upon a basic set of datatypes (binary, boolean, char, date, number, string, time) commonly used in many programming environments. An extensive list of intrinsic datatypes includes derivations of the intrinsic datatypes, including specializations of numbers, dates, and strings. User-defined datatypes may be defined by specifying a base datatype, scale parameters or an enumeration of values, and a lexical format.

2.10.2.3 Documentation

Definitions provide for accompanying documentation through the `intro` and `explain` elements. Permitted within these two element types is a collection of familiar and easy to use HTML element types. Anybody who writes HTML today will be able to write SOX documentation. The importance of the embedded documentation technique, or *Literate Programming* [63], must not be overlooked. In the right hands, this technique can be used for design, implementation, and testing in both rapid prototyping and large-scale development projects.

2.10.2.4 Inheritance

In SOX, element types may inherit their content models and attribute definitions directly from another named element type. An element type may also inherit and extend an attribute list. Specialization of attribute definitions allows refinement and restriction of attribute datatype, enumeration list and default value. Additionally, an attribute value may be defined to be inherited from the identically named attribute of a parent or older ancestor element. Thus, for example, namespaces can be inherited from superordinate elements.

2.10.2.5 Namespace support

The SOX namespace is fully and precisely defined. *Objects* from any identifiable namespace may be used in building a SOX document. That is, any element, attribute, datatype, enumeration, entity, interface, notation, parameter, or processing instruction may be imported from any namespace.

2.10.2.6 XML syntax and validation

A SOX document is a valid XML document, according to the SOX DTD. The designer of a schema, or schemographer, is free to employ the same XML tools used for traditional XML documents. This means that a SOX document can be processed by a validating XML parser, formatted according to an XSL stylesheet, and managed by any DOM-compliant application.

2.10.3 SOX – A Detailed Example

The example presented here is the SOX version of the previously used car register example (see chapter 1.3.3.1). In this example I will particularly show how accompanying documentation is provided in the SOX documents. The rules for including documentation in SOX documents are fairly simple:

- `h1` element at top of document is required.
- `h2` and `h3` elements may appear as children of `schema` elements.
- `intro` elements may appear immediately following an `h2` or `h3`.
- `h4`, `h5`, `h6`, and very large subset of HTML element types may appear as children of `intro` elements.
- An `explain` element may appear inside any *defining element.*
- `title, synopsis, help` and very large subset of HTML element types may appear as children of `explain`.

Annotated Example

```
<schema name="Car Register"
namespace="http://www.dummysite.com/carregister.xml">
<h1>Car Register Document Type</h1>
```

Every SOX document begins with the *root* element `schema`, and a top-level *heading* (`h1`) that provides a title for the SOX document. The `schema` element may be used to establish the *namespace identifier* for a SOX document.

```
<h2>Definitions</h2>
<intro>
    <p>This is a SOX document describing a simple Car Register</p>
</intro>
```

Lower-order headings with `intro` elements may be interspersed among the SOX document's defining elements to provide bridging titles and an *introduction.*

```
<elementtype name="Car">
```

I define an *element type* whose *name* is "Car".

```
    <explain>
        <title>Car Description</title>
        <synopsis>A simple description of a car</synopsis>
        <help>
            <p>
                Each instance of this element represent a Car that is stored
                in the Car Registrer.
            </p>
        </help>
        <p>A Car consists of four required fields including the Owner.</p>
    </explain>
```

I provide an explanation of this Car element `type`.

```
    <model>
        <sequence>
            <element name="Registration Number"/>
            <element name="Make"/>
            <element name="Model"/>
            <element name="Owner"/>
        </sequence>
    </model>
</elementtype>
```

The Cars *content model* is a *sequence* of four subordinate elements. As can be seen in the following fragment, the model of the first two of these elements simply contain text *strings.*

```
<h3>Car fields</h3>

<elementtype name="Registration Number">
    <model> <string/> </model>
</elementtype>
<elementtype name="Make">
    <model> <string/> </model>
</elementtype>
```

```
<elementtype name="Model">
         <model> <string datatype="number"/>< /model>
</elementtype>
```
The `Model` element's `string` content specifies its `datatype` attribute to be `number`. That means that the element value must be a number.

The `Owner` element follows precisely the same principles as the `Car` element.

```
<elementtype name="Owner">
    <explain>
         <title>Owner Description</title>
         <synopsis>A simple description of the owner of a car</synopsis>
         <help>
              <p>
                   Each instance of this element represent the person that owns a
                   car.
              </p>
         </help>
         <p>An Owner consists of four required fields.</p>
    </explain>
    <model>
        <sequence>
            <element name="Name"/>
            <element name="Address"/>
            <element name="Zip Code"/>
            <element name="City"/>
        </sequence>
    </model>
</elementtype>
<elementtype name="Name">
    <model> <string/> </model>
</elementtype>
<elementtype name="Address">
    <model> <string/> </model>
</elementtype>
<elementtype name="Zip Code">
    <model> <string datatype="number"/> </model>
</elementtype>
<elementtype name="City">
    <model> <string/> </model>
</elementtype>
</schema>
```

2.11 Vector Markup Language (VML)

Unlike the other specifications listed in this section, the Vector Markup Language (VML) specification does not try to extend the XML specification by adding new "functionality", but instead tries to use it to build an XML application language. Two other XML application languages have previously been constructed: the Mathematical Markup Language (Math-ML) [29] and the Chemical Markup Language [54].

The VML specification is a joint Autodesk, Hewlett-Packard, Macromedia, Microsoft, and Visio submission to the W3C.

VML defines a format for the encoding of vector information together with additional markup to describe how that information may be displayed and edited. VML supports the markup of vector graphic information in the same way that HTML supports the markup of textual information. Within VML the content is composed of paths described using connected lines and curves. The markup gives semantic and presentation information for the paths. VML uses Cascading Style Sheets Level 2 (CSS2) [3], in the same way as HTML to determine the layout of the vector graphics that it contains.

"The Web community has been asking for a high-quality, easy-to-use 2-D vector graphics standard for some time. VML meets their needs with faster graphics downloading for end users and easier graphics editing and manipulation for HTML authors and designers. VML will be a key specification in our future platforms and applications."

-- David Cole, vice president of the Web client and consumer experience division at Microsoft.

2.11.1 Requirements

Many requirements guided the design of VML. The most crucial are listed below in order of importance.

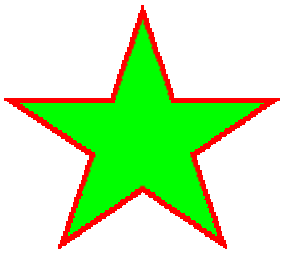
1. Retain the information required for further editing of VML. This requirement has the important consequence that VML must be extensible - it is inconceivable that VML meets the requirements of all editing applications, therefore it must be possible for every application to add the required editing data specific to that application.
2. Support interchange of data between applications. One application must be able to read and edit the data of another application, even though the first requirement means that, potentially, both applications will add application specific data.
3. Use the existing mechanisms of HTML and CSS - this facilitates implementation of VML and ensures that implementations can reuse existing code and techniques.
4. Be backward compatible with existing user agents. VML adoption will be inhibited unless it is possible to produce VML, which works with existing Web browsers. VML has special provisions to allow alternate bitmap representations of graphics for backward compatibility.
5. Provide efficient representations of vector graphics. Textual representations tend to be verbose. VML addresses this by defining a compact representation of path elements and by following a design principle of using concise names for frequently used attributes and more verbose names for less frequently used attributes.
6. Allow the implementation of subsets where an application does not require the full functionality of VML. Normally a viewer will implement the full specification, however editors should be able to implement only the subset required for their own data.
7. Support hand editing. This leads to a design principle that the structure of the graphic be obvious and that the syntax be familiar to HTML programmers - effectively the same as requirement (3).
8. VML should support scripting, including the requirements of animation. This, again, leads to a desire for the structure of VML to match the structure of the graphics. It also leads to the use of types within VML attributes which are appropriate to animation - for example 2D coordinates are defined as single attributes "x, y" rather than pairs of attributes.

2.11.2 Structure

The overall structure of VML may be summarized by the XML definitions of the two primary elements - `shape` and `group`.

A `shape` element is used to define a visible vector graphic element. Most shapes have a *path* definition; a sequence of straight lines and cubic Bézier curves which defines an outline. The outline may be *stroked*, as specified by attributes on the shape and the `stroke` sub-element. It may also be *filled*, under the control of shape attributes and the `fill` sub-element. Additional sub-elements support raster (bitmap) images, more advanced transformations of the path and text drawn on top of the shape.

Below is an example of a simple shape and its VML representation. I will not go into detail about how to interpret the VML code, instead I refer the interested reader to the VML specification [12].



```
<v:shape style='top: 0; left: 0; width: 250; height: 250'
stroke="true" strokecolor="red" strokeweight="2" fill="true"
fillcolor="green" coordorigin="0 0" coordsize="175 175">
<v:path v="m 8,65
l 72,65,92,11,112,65,174,65,122,100,142,155,92,121,42,155,60,100
x e"/>
</v:shape>
```

Figure 7: A simple shape and its VML representation.

A `group` element is used to group together several shapes so that they may be transformed together as one unit. In addition VML defines several auxiliary top-level elements to help make the editing and representation of complex graphical information more compact and convenient. The `shapetype` element is used to define a prototype definition of a shape. A `shape` element may reference a `shapetype` in order to instantiate several copies of the same shape. Several predefined shapes may be used as convenient

alternatives to explicitly declaring a shape element with a path. These predefined shapes are `line`, `polyline`, `curve`, `rect`, `roundrect`, `oval`, `arc`, and `image`.

2.11.3 Use of CSS

The style attribute in the shapes uses the syntax described in “Visual rendering model” in CSS2. The positioning may be absolute or relative unless the shape is within a group, in which case it must be absolute (relative to the top left of the parent group). The z order of the elements within the group is from the first (lowest) to the last (highest); i.e., later elements obscure earlier elements. The elements establish no relative position, hence the restriction to use of absolute positioning.

The VML `shape` and `group` elements participate fully in the CSS2 visual rendering model. In addition to standard CSS layout the VML elements may also be rotated or flipped. Each element also establishes a coordinate space for its content, this allows scaling of the content with respect to the containing elements.

2.11.4 Future Support

VML will be supported broadly by Autodesk, Hewlett-Packard, Macromedia, Microsoft, and Visio in future versions of their products. Microsoft plans support of VML in Microsoft Internet Explorer 5.0, the Windows operating system and the next version of Microsoft Office. VML support in the next version of Office will allow users to save Office Art graphics as editable elements in their HTML pages for delivery via the Web. VML will preserve the full fidelity of Office Art objects and allow "round tripping"; that is, the HTML file can be opened and edited back in an Office application with no loss of quality.

Part 3: Using XML

3.1 A Common XML Communication Architecture

Now that we have a basic understanding of XML and associated standards, how are they supposed to work in reality? The basic structure of future XML applications will often be developed using a *three-tier* architecture. This approach has in recent years become very popular when developing Web applications. It consists of three components (hence the name three-tier); one or more data sources, a server located somewhere in the Internet, and clients connecting to that server.

Many companies have been closely following announcements from Microsoft and Netscape about the XML capabilities in their next-generation browsers. Using Internet Explorer 4.0 or 5.0's built-in XML parser, they have prototyped building very "thin", highly dynamic user interfaces leveraging Dynamic HTML. To minimize the number of trips to the Web server and to avoid refreshing the entire browser page on each request, these developers use JavaScript to request new XML "datagrams" from the middle-tier via HTTP. The browser's built-in XML parser receives the stream of tagged data from the middle-tier and exposes the Document Object Model interface on the resulting "tree of data" so that the developers' JavaScript can inspect what data has "arrived" and update the user interface to reflect any changes.

3.1.1 Three-Tier Application Architecture

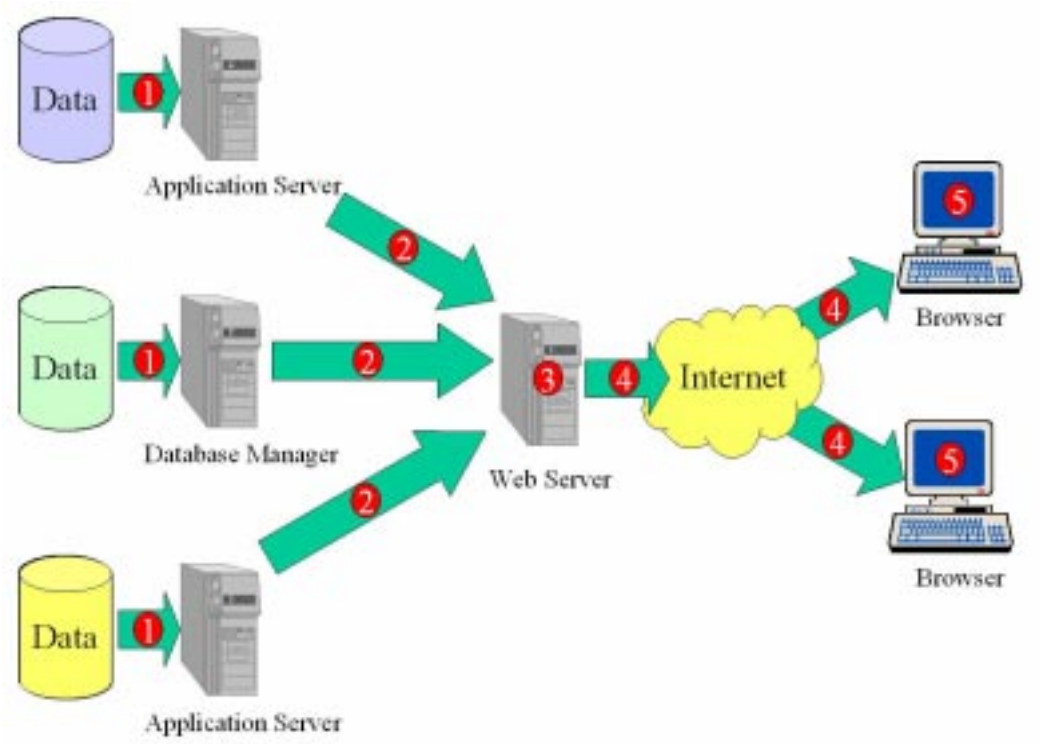


Figure 8: Three-Tier Application Architecture

All the components in the architecture are described below:

❶ The data used in the application can be stored using different media, formats, and platforms. What they all have in common is that they all are accessed through specific data-managers (database managers, application servers, and so on).

❷ The communication with the data-managers are today handled using several different techniques; ActiveX components, DCOM, COM, CORBA, JDBC, to only mention a few. The problem is that the big variety of communication techniques makes it hard for different data-managers to interact, not to mention the problems for the developers trying to make it all work. It would be much better

if they all could “talk the same language”. Work is currently being done to enable the different data managers to communicate using XML directly.

3. The data is gathered from the different data managers in a server that is often referred to as the “*middleware*” of the application. This middleware can among many other things consist of; Active Server Pages (ASP), Java, Java-Scripts, and the list goes on. The purpose of the middleware is to collect data, process it, and send it to the client in an appropriate format.
4. Today that format is often HTML but XML would in the future be a better choice because the data can then be further processed on the client. The protocol used between the browser and the server is in most cases HTTP.
5. The clients receive the data and present it in a Web browser. Soon the major Web browsers will support XML directly, but currently they only handle HTML and DHTML.

3.2 Support for XML (and associated standards)

In a few short months XML has attracted more than its share of industry backing and media hype, but XML browsing and editing tools are still few, and many XML developers predict that they will not appear on the market for some time. This is a big problem when trying to evaluate, test, and use the standards.

Low-level tools, parsers that read in XML documents and present them to an application in standardized form, have now been available for some time and many of these tools are free. The many parsers available include efforts from corporate workshops like IBM alphaWorks, Microsoft, DataChannel, and Microstar, and from independent XML and SGML developers like James Clark and Tim Bray. Parsers are available in a number of programming languages, including Java, C, and C++, and many are capable of validating XML documents.

Browsers that support specific XML applications, like MathML (the Amaya browser [33]) and Chemical Markup Language (the Jumbo browser [55]) are also available. A limited amount of support, requiring scripting, is also available in Microsoft's Internet Explorer 4.0. A variety of tools for creating XML, transforming it into other markup (typically SGML or HTML), and using it for specific applications are also arriving slowly.

Most of the tools available today are very limited and don’t exploit all the possibilities in the standards that they support. Neither do they provide much transparency for XML. If users have the bad fortune to receive an XML document in the current mainstream (Netscape or Microsoft) browsers, they may see the markup in all of its glory spread across their browser window, they may get a blank screen, or they may be asked if they want to save the file to their hard drives, depending on how the browser is configured. XML documents cannot be rendered with the existing Cascading Style Sheets (CSS) functionality already, though incompletely, built into the browsers. As a result, XML is not yet a reasonable choice for sending information to end users, thus they need either a transformation to HTML or a separate viewing application.

This may be acceptable for a narrow range of applications that need a separate viewing application anyway, and some organizations may not mind incurring the processing costs of transforming XML documents to HTML. It is not acceptable for organizations or individuals that expect to be able to publish in the traditional style of the Web, uploading files (the document plus possibly a style sheet) to a server without the need for transformations. Applications that publish information from programs are also unlikely to switch their output to XML if the XML will still need to undergo additional transformation before becoming usable to the average Web browser.

This is one of biggest current problems that is holding back widespread usage of XML. Until there exist browsers that fully incorporate the major parts of XML (and XSL) it is unlikely that it will be used. There is however much development going on to create such products. Both Microsoft and Netscape have expressed a desire to incorporate XML and XSL in their Web Browsers. Netscape's Mozilla open-source project with its community of developers is adding XML support, including the W3C DOM to Communicator 5.0. Microsoft, meanwhile, made its first try at a DOM implementation, based on the W3C's work, in the Internet Explorer 5.0 beta 1 release.

Although both Microsoft and Netscape have agreed to adopt XML, each company seems to be maneuvering to make XML suit its own proprietary needs. Microsoft is using XML for its new Channel Definition Format for "push" media. Netscape will integrate the new language into its upcoming Web-

design software suite, code-named Gemini, hoping XML and Gemini will form the centerpiece of the network computer that Apple and Sun are building as an alternative to today's dominant Intel and Microsoft Windows PC standard.

The trend to "companify" the standards is made even clearer by the different companies presenting their own XML-to-HTML formatting engines that completely ignore CSS and XSL. The common justification for these products is that they can integrate data from databases into Web pages.

The pricing of these products is another limiting factor; Interleaf and WebMethods have document management tools in the five-figure dollar range. Most of the other XML products are in the four-figure region. By contrast, most new HTML products lie between $100 and $300. Clearly, these companies don't see XML as a mass-market technology. If XML is going to take off, the freeware community is going to have to do the heavy lifting. Otherwise, at these prices XML is limited to the same markets as SGML.

3.2.1 XML Parsers

As I previously mentioned, there are now a number of XML parsers available. An XML parser is software solely dedicated to reading and interpreting XML documents to enable further processing by other applications. They do this by giving the developer access to classes that can process XML documents. These classes can then be called by other classes in an application through a common interface (almost all parsers support the DOM Level 1 interface).

The most popular language to write XML parsers in is Java even though there are also parsers available in other languages.

Two pairs of traits distinguish the different XML Parsers:

- Whether they are *validating* (checks DTD) or *non-validating* (checks for well-formedness, no DTD checking).
- Whether they are lightweight and therefore intended for use in applets or whether they are best suited for full-fledged applications.

Here is a small list of Java XML parsers that are available from different companies or independent developers. A most interesting comparison between different XML parser can be found in [58].

Name	Company/Developer	Validating	Lightweight	URL
Aelfred	Microstar	No	Yes	http://www.microstar.com/aelfred.html
XP	James Clark	No	No	http://www.jclark.com/xml/xp/index.html
Larval	Tim Bray	Yes	Yes	http://www.textuality.com/Lark/
Microsoft XML Parser for Java (XJ2)	Microsoft And DataChannel	Yes	Yes	http://www.datachannel.com/xml/developers/parser.shtml
Java Project X Parser	JavaSoft	Yes	Unknown	http://developer.javasoft.com/developer/earlyAccess/xml/
XML Parser for Java (XML4J)	IBM AlphaWorks	Yes	No	http://www.alphaworks.ibm.com/formula/xml

3.2.2 Future Support

As I have previously stated, the only thing that holds back XML is the lack of support and tools available from the leading software companies. However this will soon change with Microsoft leading the charge. Microsoft is actively involved in defining the emerging XML standard and will continue to implement XML as defined by the W3C. As a co-founder of the W3C's XML Working Group, Microsoft enjoys broad support for its efforts from many participants in the W3C process.

Microsoft wants to use XML to obviate some of the advantages sought by their competitors through the emerging Enterprise JavaBeans (EJB) standard [48]. EJB allows the sharing of applications and components across distributed, heterogeneous environments.

"XML holds great promise for customers who need universal data communications with anyone, anywhere."
-- J. Allard, general manager of Windows DNA infrastructure at Microsoft

Right now, Microsoft is aiming to prime the programmer pump on XML; they have gathered 6,000 developers to support the development of XML within their product line. Microsoft has announced that they will support the following key features in the next update to the Windows operating system and its Internet Explorer browsing software:

- **Direct viewing of XML.** The Microsoft XML implementation lets users view XML using XSL or Cascading Style Sheets with their Web browser, just as they view HTML documents.
- **High-performance, validating XML engine.** The XML engine first encountered in Internet Explorer 4.0 will be substantially enhanced and fully support W3C XML 1.0 and XML Namespaces, which lets developers qualify element names uniquely on the Web and thus avoid conflicts between elements with the same name. Native XML support in Windows means that developers can count on the full XML processing capabilities being present to read and manipulate the data they move between their applications and components.
- **Extensible Style Language (XSL) support.** With the Microsoft XSL processor, based on the latest W3C Working Draft, developers can apply style sheets to XML data and display the data in a dynamic and flexible way that can be easily customized. The querying capabilities of the Microsoft XSL processor also allow developers to programmatically find and extract information within an XML data set on the client or the server.
- **XML Schemas.** Schemas, as previously described, define the rules of an XML document, including element names and data types, which elements can appear in combination, and which attributes are available for each element. In order to enable multitier applications, Microsoft will be releasing a technology preview for XML Schema based on the Schema for Object-oriented XML (SOX) submission to the W3C XML working group.
- **Server-side XML.** Server-side XML processing allows XML to be used as a standard means of passing data between multiple distributed application servers - even across operating system boundaries.
- **XML document object model (DOM).** The DOM is a standard object application-programming interface that gives developers programmatic control of XML document content, structure, formats and more. The Microsoft XML implementation includes full support for the W3C XML DOM Recommendation and is accessible from Visual Basic- or Java-script, the Visual Basic development system, C++, and other languages.

In addition to these innovations, Microsoft is using XML in its applications software. For example, the next major release of the Microsoft productivity suite, Microsoft Office 2000 (Office 2K), elevates HTML to a companion file format and uses XML to store additional document information. By using XML in this way, Office 2K users can save documents as Web pages and then later return these documents to their original Office state for editing.

Microsoft will also include support for VML in IE5.0. The advisability of this can be questioned, since VML has received nobody's blessing and in fact the W3C plans to combine several proposals into a "Single Vector Graphics" (SVG) facility. Microsoft's position is that VML is going to be the Office 2K vector graphics format (nobody disputes that Office needs such a thing) and thus the IE5.0 browser needs to support it so that it can display Office documents.

Looking at the Microsoft Office 2K implementation, though, it's basically Microsoft saving its own relevant information through XML data, rather than creating a vendor neutral solution. This is a deliberate approach by Microsoft; sure you can save into HTML/XML, but there's only going to be one browser on the planet that will be able to open it up and do anything useful with it. Microsoft does not see XML as an interchange format for vendor neutral solutions; they see it as a database format to augment their current products.

With Microsoft so fully embracing XML there is no risk of the standard falling into oblivion. As stated above, there is however a danger that Microsoft will try to make the standard "their own" and thus severely limiting the openness of the standard.

Another trend that is becoming more and more apparent is that the traditional database companies (Oracle, Sybase, and more) are working to incorporate XML into their database engines. Oracle will have their first implementation of XML services available in their Oracle8*i* database manager, and the others are not far behind.

3.3 An Example of XML and XSL

To show a small example of what currently can be done with XML and XSL I have constructed a small application that implements a car record. The application consists of two parts, an XML document and a XSL stylesheet.

The XML file [http://www.student.nada.kth.se/~d94-pno/exjobb/carregister/Car.xml] is the "database" of the application; here is all the information about different cars stored using the XML v1.0 format. The XSL file [http://www.student.nada.kth.se/~d94-pno/exjobb/carregister/car.xsl] describes rules to present the data stored in the XML file using the XSL v1.0 format. This XSL-file is parsed using the Microsoft XSL Command-Line parser to construct a dynamic HTML page. The dynamic HTML page uses the Microsoft Internet Explorer 4.0 (IE4.0) XML Object Model and JavaScript to make the page interactive. A strange fact is that even though the application uses the IE4 Object Model, it only works under Internet Explorer 5.0 (IE5.0). Obviously IE4.0 isn't quite as compatible with XML as Microsoft claims. Another limitation is that the IE4.0 XML Object Model don't allow changes to be written to disk, so all changes are only stored locally and disappear if you press the refresh button in your browser.

The application itself can be found at:
[http://www.student.nada.kth.se/~d94-pno/exjobb/carregister/xmlexample.html].
Observe that it only works with IE5.0. The application itself is very simple, you can view all the records by clicking on the Next and Previous buttons. You can delete records, except when there is only one record left; this is due to the fact that the IE4.0 object model don't seem to handle an empty XML document very well. You can also add records; the information currently in the input textboxes is stored as the new record.

Figure 9: This is a picture of the GUI in the XML example, the real application can be found at: http://www.student.nada.kth.se/~d94-pno/exjobb/carregister/xmlexample.html

3.4 XML and Electronic Transactions

Companies have today put great effort into constructing computer applications to help them in their business processes. While this has resulted in significant improvements in efficiency, that efficiency has not been extended to external processes. By external processes I mean processes that involve interchange between applications or business processes at different companies. Companies have in many cases created islands of automation that are isolated from their suppliers, trading partners, and customers. Electronic Data

Interchange (EDI) has been heralded as the solution to this problem. EDI is the standardization of data-exchange between heterogeneous systems to support transactions.

3.4.1 EDI

EDI is commonly defined as the application-to-application transfer of business documents between computers. Many businesses choose EDI as a fast, inexpensive, and safe method of sending purchase orders, invoices, shipping notices, and other frequently used business documents.

EDI is quite different from sending electronic mail messages or sharing files through a network, a modem, or a bulletin board. The straight transfer of computer files requires that the computer applications of both the sender and receiver (referred to as "trading partners") agree upon the format of the document. The sender must use an application that creates a file format identical to the receiver's computer application.

When you use EDI, it's not necessary for the trading partners to have identical document processing systems. When the sender sends a document, EDI translation software can convert the proprietary format into an agreed upon standard. When the receiver receives the document, his EDI translation software automatically changes the standard format into the proprietary format of his document processing software.

The idea behind EDI is a good one, unfortunately it hasn't worked out so well in reality. One of the major problems with the current implementations of EDI is that they often require a unique solution for each pair of trading partners, making EDI costly and time-consuming to implement.

Another problem with traditional EDI is that it is based on the use of rigid transaction sets with business rules embedded in them. These transaction sets are defined by standards bodies such as the United Nations Standards Messages Directory for Electronic Data Interchange for Administration, Commerce, and Transport (UN/EDIFACT) [30] and American National Standards Institute's Accredited Standards Committee X12 sub-group (ANSI X12) [32]. Transaction sets define the fields, the order of these fields, and the length of the fields. Along with these transactions sets are business rules, which in EDI-language are referred to as "*implementation guidelines*". So why is a rigid set of transaction sets not such a good idea?

"It is not the strongest species that survive, nor the most intelligent, but the one most responsive to change."
-- Charles Darwin

A fixed transaction set prevents companies from evolving by adding new services and products or changing business processes. The bodies that make the standard transaction sets are ill equipped to keep up with the rapid pace of change in the various business environments they impact. It is also very hard, if not impossible, to develop a one-size-fits-all solution.

"The big problem is that the formal EDI standards that exist today, such as UN/EDIFACT and ANSI X12, were developed twenty-five years ago. New business practices, the development of global economies, and advancements in computer technologies are just several of the factors that have made those standards unworkable and unimplementable for many organizations (only 100,000 companies worldwide today use EDI, which is rather surprising for standards and technologies that have been in play for a quarter of a century). For one thing, the existing EDI standards only accommodate point-to-point transactions. For another, as with many "standards," actual implementations are typically customized to suit specific application requirements. With EDI, such customizations are embodied in Implementation Conventions (ICs). In many instances, ICs have become barriers to effective data interchange. Different ICs on the buyer and seller sides of the transaction make for data that's not interoperable without translation or conversion of some sort."
--Mary Fletcher Laplante, Director, Document Software Strategies Group.

3.4.2 EDI using XML

By using XML to implement EDI many of the old problems are eliminated. XML maintains the content and structure, but separates the business rules from the data. By focusing on exchanging data content and structure, trading partners can apply their own business rules. Using XML it is also very easy to extend the communication to support new business processes, EDI will no longer be limited to rigid standards.

"XML supports the development of dynamic repositories; no longer do EDI applications have to be "shoe-horned" into fixed element dictionaries or templates."
--Mary Fletcher Laplante, Director, Document Software Strategies Group.

When I say that EDI using XML is not limited to rigid standards that doesn't mean that it doesn't have to conform to ANY standards. Trading partners still need to agree upon the format and contents of the messages to be able to interpret them. What XML provides is primarily:

- **Self-explaining syntax**: Since the elements in XML are discoverable using document type definitions, the elements used to describe a supplier's product, its pricing, or other attributes can be gleaned without first having to agree upon a single format beforehand. Previous visions of EDI could not use this kind of ad hoc partnership approach. At best, an industry might define a set of EDI templates or forms for specific transaction types. Generally, these would be established by the largest company in a supply chain as a de facto set of transaction standards and datatypes.
- **Modularity**: The XML – EDI messages can be constructed using a combination of several standardized modules. It is possible to provide a number of standardized and publicly accepted building blocks that can be used to construct more complex EDI messages. This is quite different from the current implementations where all the functionally has to be included in the messages from the beginning, resulting in that people are adding all kinds of messages in the standards because they MAY be used in the future.
- **Extendibility**: Since the EDI standard using XML is no longer under the eye of standardization committees, such as UN/EDIFACT and ANSI X12, it is much easier to add new support and functionality in the EDI messages.
- **Presentation**: The XML – EDI messages can be presented directly to the user using the XSL specification (when and if such support becomes available in the browsers).
- **Transformation**: The XML – EDI messages can be processed, transformed and analyzed using the proposed XML-QL standardization.

The fact that XML documents can easily be distributed using the Internet is another major advantage. This combined with the fact that XML is self-explaining provides the entire framework for what the XML/EDI group calls a new "*supply web*". Earlier many EDI implementations only worked within their own Intranets. With XML this is no longer the case since XML provides connectivity through the Internet. All applications are then able to communicate and exchange data, thus the old point-to-point solutions are history. To many people XML-EDI is also what they call a 'politically correct' way to merge ANSI X12 (the US standard) and EDIFACT (the world standard).

"The goal is to establish the standard for future EDI that is open and accessible to all vendors and end users alike. More importantly, it is not only extendible into the future, but also adaptable to incorporate new technologies."
--David Webber, a founding member of the Internet-based XML/EDI Group.

3.4.2.1 EDI using XML – Example

Consider a small, Web-based retailer that stocks inventory from a major manufacturer. For years the company has been ordering merchandise from the manufacturer using a human-readable purchase order such as that in Listing 1, for 1000 fuzzy dice. Now the manufacturer is demanding that the retailer begin using EDI for transactions or else it will add a surcharge to every order to offset the cost of handling manual transactions. The small retailer can't find a more lenient supplier, so it has little choice.

Listing 1: Plain purchase order

P.O. Number:	003429
Date:	1 December 1998
Order Contact:	John Johnson
Company:	Internet Retailer Inc.
Address:	123 Via Way, Milwaukee WI, 53202

Qty	Part No.	Description	Unit Price	Total
100	CO633	Fuzzy Dice	$1.23	$123.00

Unfortunately, EDI compliance means the retailer will now have to transmit its purchase orders in a format specified in one of the several standard sets for EDI format, the most widely used being ANSI X12 and UN/EDIFACT. Listing 2 shows part of an ANSI X12 version of the purchase order in Listing 1. It's possible that the retailer's current accounting software supports EDI transactions, but not likely. It will probably have to either convert to software that does, purchase software that can make the translation (if available), or contract with a third party to convert the data on an ongoing basis.

Listing 2: Fragment of ANSI X12 transaction set corresponding to Listing 1

```
ST*850*12345
BEG*00*SA*3429**981201
N1*BY*Internet Retailer Inc.*91*RET8999
N1*ST*Internet Retailer Inc.
N3*123 Via Way
N4*Milwaukee*WI*53202
PER*OC*John Johnson
PO1**100*EA*1.23*WE*MG*CO633
SE*9*12345
```

So, would there be any advantage, for the retailer or the manufacturer, to using an XML format instead? Perhaps the retailer's accounting software is tailored to Internet commerce, and already uses XML formats. In this case, there are utilities for converting one data type definition (DTD) to another, and the conversion to EDI might be accomplished by a generic tool. The resulting transmission would certainly be more comprehensible to humans, and perhaps even feasible for an employee with a text editor or XML editor to generate. An example of how the sample purchase order might look in an XML rendering of X12 is given in Listing 3. This example is based on the ongoing work of the XML/EDI group. One disadvantage with using XML, as can be seen below, is that the size of the EDI messages gets a lot bigger than the X12 messages. In this example, the XML message is about 8 times bigger than the X12 message.

Listing 3: XML document analog to the X12 transaction set in Listing 2

```
<?XML version="1.0" encoding="UTF-8"?>
<PurchaseOrder Version="4010">
   <PurchaseOrderHeader>
      <TransactionSetHeader X12.ID="850">
         <TransactionSetIDCode code="850"/>
         <TransactionSetControlNumber>12345</TransactionSetControlNumber>
      </TransactionSetHeader>
      <BeginningSegment>
         <PurposeTypeCode Code="00 Original"/>
         <OrderTypeCode Code="SA Stand-alone Order"/>
         <PurchaseOrderNumber>RET8999</PurchaseOrderNumber>
         <PurchaseOrderDate>19981201</PurchaseOrderDate>
      </BeginningSegment>
      <AdminCommunicationsContact>
         <ContactFunctionCode Code="OC Order Contact"/>
         <ContactName>John Johnson</ContactName>
      </AdminCommunicationsContact>
   </PurchaseOrderHeader>
   <PurchaseOrderDetail>
      <Name1InformationLOOP>
         <Name>
            <EntityIdentifierCode Code="BY Buying Party"/>
            <EntityName>Internet Retailer Inc.</EntityName>
            <IdentificationCodeQualifier Code="91 Assigned by Seller"/>
            <IdentificationCode>RET8999</IdentificationCode>
         </Name>
         <Name>
            <EntityIdentifierCode Code="ST Ship To"/>
            <EntityName>Internet Retailer Inc.</EntityName>
         </Name>
         <AddressInformation>123 Via Way</AddressInformation>
         <GeographicLocation>
            <CityName>Milwaukee</CityName>
            <StateProvinceCode>WI</StateProvinceCode>
            <PostalCode>53202</PostalCode>
         </GeographicLocation>
```

```
      </Name1InformationLOOP>
      <BaselineItemData>
         <QuantityOrdered>100</QuantityOrdered>
         <Unit Code="EA Each"/>
         <UnitPrice>1.23</UnitPrice>
         <PriceBasis Code="WE Wholesale Price per Each"/>
         <ProductIDQualifier Code="MG Manufacturer Part Number"/>
         <ProductID Description="Fuzzy Dice">CO633</ProductID>
      </BaselineItemData>
   </PurchaseOrderDetail>
</PurchaseOrder>
```

3.4.2.2 XML-EDI Tags

Currently a big debate is going on between the different XML-EDI groups on what the structure of the XML tags should be. Some advocate the use of the full description in the tag and others the use of the ANSI element number as the tag name. Others want to use a unique number that would include the standard version of the directory, the segment it resides in and the data-element it represents. In the view of the European Electronic Messaging Associations (EEMA) EDI work group, there should be a sequential numbering of the tags for XML-EDI; this would be an identifying code only, starting with a letter. This would provide a neutral starting point for all old standards and would be most efficient in size.

3.4.2.3 Repositories

XML-EDI incorporates the concept of a global dynamic repository. The repository is proposed to act like a dictionary that defines the semantics of terms used in a transaction. That repository would also be used to hold common Document Type Definitions under the XML/EDI proposal.

"The repository is a location where shared Internet directories are stored and where users can manually or automatically look up the meaning and definition of XML-EDI Tags. The repository is in fact the semantic foundation for the business transactions."
--XML/EDI Group.

XML repositories will provide a means for industries to store on the World Wide Web the document-type definitions that identify the data elements and their relationships exchanged among parties doing business electronically. Repositories will also contain logic components, such as Java applets, template scripts, forms and object definitions needed to process message components.

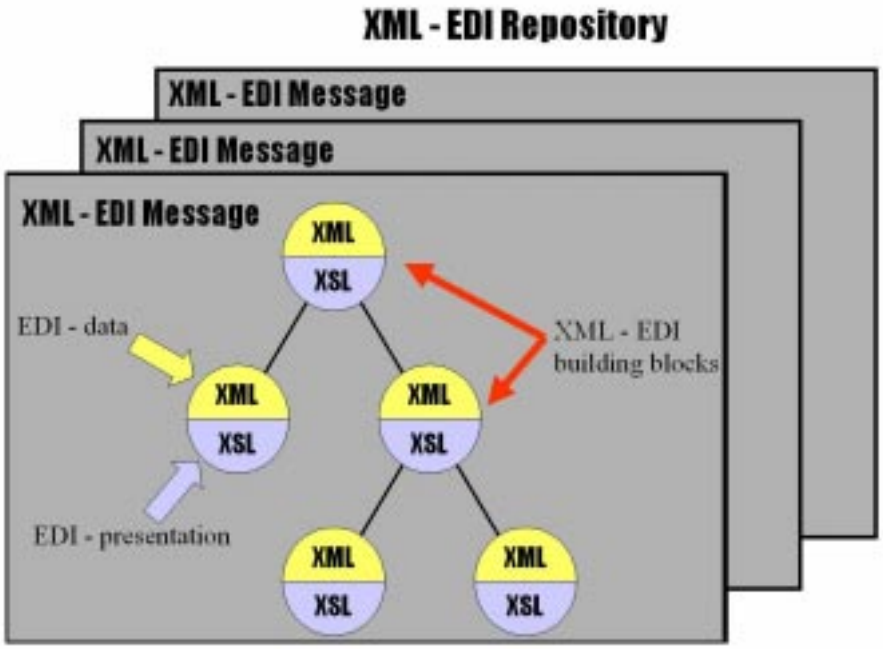


Figure 10: Shows the global repository of standardized EDI messages that themselves are made up by standardized XML – EDI building blocks. The building blocks contain both an XML document containing the structure (DTD) and data, and an XSL stylesheet to display that document.

With common registration procedures for these components, repositories will act as global libraries, and enable industry groups, government agencies, and businesses of all sizes to make their preferred message formats widely available to current and potential trading partners.

"XML repositories will ultimately redefine not only how companies do business but also how they implement business applications. The key is in providing an open architecture to facilitate the wholesale shift of business to the Web."
--Bruce Peat, chair of the XML/EDI Group.

It is anticipated that fully functional global repositories will evolve in phases:

- Phase 1 - Limited Intranet implementation, proof of concept, with manual Web interface.
- Phase 2 - Definition of basic API allows Extranet implementations between specific partners.
- Phase 3 - Definition of full API and available repository products allows full implementations.
- Phase 4 - DNS style service established and standards bodies adopt support for API and long term maintenance and alignment.

Example: An XML-EDI transaction using a global repository

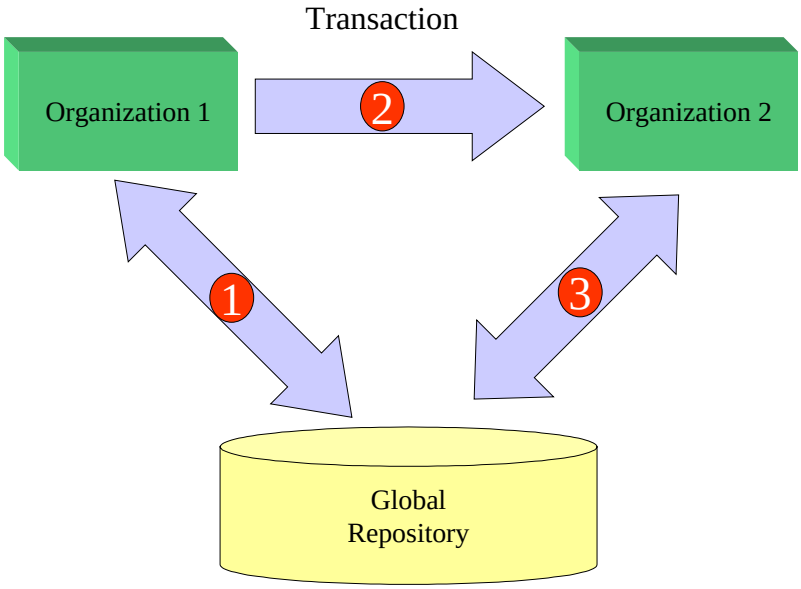


Figure 11: XML/EDI Transaction Schematics

❶ A user or agent in Organization 1 queries the Global Repository for the structure (DTD) of those common business objects that are to be passed to a trading partner - Organization 2.

❷ The business objects are passed to Organization 2, using XML documents structured according to the DTDs fetched from the Global Repository.

❸ The receiver maps the received data in to the organization's application system. The mapping is received from the Global Repository.

The intent of the Global Repository is to be a dynamic mechanism that responds through an open Application Programming Interface (API).

3.4.2.4 Current and future support

Currently there is much work going on in this area. The XML/EDI group [http://www.geocities.com/WallStreet/Floor/5815/] is working to define XML namespaces and DTDs equal to the EDIFACT standard to support commercial usage of EDI over XML. Even though most organizations agree that EDI using XML is a good idea some are still skeptical. This is due to the fact that they think that EDI is too complex to be dealt with easily. In the health care industry alone, there are 400 different formats for a claim. One of the key issues will be how compatible the new implementations of EDI will be to the old ones; currently there is no answer to this question.

Enterprise resource planning (ERP) heavyweights such as PeopleSoft Inc., Oracle Corp., and SAP AG plan to incorporate XML syntax into products to help companies lower the cost and labor involved in exchanging data with business partners. Separately, third-party software vendors are pushing XML to enable data exchange over the Web between separate financial systems.

One such vendor is webMethods who claims to have the first and currently only XML-based e-commerce solution in their Business to Business (B2B) product. B2B automates the exchange of data between applications, Web sites, and legacy data sources using XML. At the heart of webMethods B2B is the Web Interface Definition Language (WIDL) [11]. Introduced in 1996, WIDL was one of the first applications of XML. WIDL provides a means of describing automated access to Web-enabled resources and enterprise applications through well-defined interfaces much like COM and CORBA. By abstracting information about Web and external application resources, applications can be protected from changes in the structure, appearance, and location of data.

3.5 Product Data

In the world today there is a great need to digitally store information about products. This information is commonly called *product data*. The term product has here a very wide range and can be anything from roads to spoons. One of the key characteristics of product data information is that it is modified (created, used, and added to) throughout the life cycle of a product. For instance, the basic shape of a building will be created by an architect (design phase), modified by structural engineers (analysis phase), realized by constructional engineers (production phase), and used in managing and maintaining the building (operations and maintenance phase).

Today there exists a number of applications and systems that handle this kind of data, the problem is as usual how to make them talk to each other. As a solution to the communication problem and to enable the inter-working of different systems, a product data integration standard has been developed. This standard has been named STEP.

So, why is there a chapter about product data and STEP in an XML review? Decerno is currently working closely together with Vägverket on an infrastructure for storing and viewing the road network in Sweden. The information about the roads will be stored in two systems called GALANT and NVDB (Nationell VägDataBas).

Decerno has developed the GALANT system to store road information for the Swedish municipalities. CAP (another Swedish software consulting firm) is currently developing NVDB and it will be used to store information about roads for all of Sweden. These two systems need to be able to communicate with each other. It has been decided that NVDB will use STEP part 21 (described below) to transfer information to and from it. But STEP has some limitations. Decerno is therefore interested in the possibility of in the future replacing STEP with XML. Is it possible and what will the advantages and disadvantages be of this possible replacement? To answer this question we first need to understand something of the STEP standard.

3.5.1 STEP

STEP originally stands for **ST**andard for the **E**xchange of **P**roduct Model Data, however the scope of the name has recently been broadened to stand for Standard for the Exchange of Industrial data. STEP is the unofficial name, the official name of the STEP standard is *ISO 10303 Industrial Automation Systems – Product Data Representations and Exchange* [23]. The work on the STEP standard was initiated in 1984, with the following objectives:

- The creation of a single international standard, covering all aspects of CAD/CAM data exchange.
- The implementation and acceptance of this standard in industry, superseding various national and de facto standards and specifications.
- The standardization of a mechanism for describing product data, throughout the life cycle of a product, and independent of any particular system.
- The separation of the description of product data from its implementation, such that the standard would not only be suitable for neutral file exchange, but also provide the basis for shared product databases, and for long-term archiving.

STEP is a very big standard, but it has three key components:

- The EXPRESS data specification language.
- Data Models

- Implementation Forms.

3.5.1.1 EXPRESS

One of the key objectives of STEP is to provide an unambiguous and computer interpretable representation of product data. This is supported through the use of the EXPRESS language [24]. EXPRESS is a data definition language that is used to represent the structure of data and any constraints that may apply to it. The structured data is usually divided into separate modules called data information models or simply *data models*. Although EXPRESS resembles some programming languages, it can not be used to define executable programs. It is used to define the data on which programs operate. EXPRESS supports:

- The definition of data entities, attributes, and relationships.
- The specification on local and global constraints on the entities.
- The collection of data definitions and constraints in separate schemata, supporting modular development of data models.

EXPRESS code can be written as text and viewed using EXPRESS-G, a simplified graphical representation of the EXPRESS models. Here is an example using both methods.

Example: Car data as EXPRESS code

```
ENTITY car;
       make   :      STRING
       model  :      STRING
       year   :      INTEGER
       owner  :      person
END_ENTITY

ENTITY person;
       first_name    :      STRING
       last_name     :      STRING
END_ENTITY
```

Example: Car data as an EXPRESS-G graphical representation

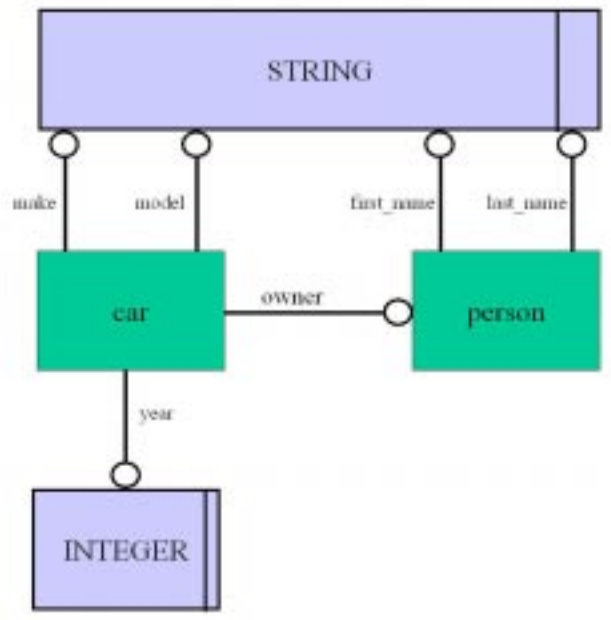


Figure 12: An Express-G representation

3.5.1.2 Data Models

STEP's data models are extremely generic and can in principle handle any kind of entity, including people, things, processes, documents and information content. The data models are, as said above, defined using the EXPRESS language. They are used to exchange and share product data. This is accomplished through the standardization of usable data models. There exist two types of standardizations:

- **Generic product data models**: That supports the requirements of **all** product data applications. It has a modular structure that supports extensibility and phased development. As a result, this generic model is published in several different parts of the STEP standard known as *Integrated Resources*.

- **Specific product data models**: To support the need for more application-specific data models the generic product models can be constrained and sub-classed to form more specific data models. These models are standardized in STEP as *Application Protocols*, which form application-specific "views" within the standard.

3.5.1.3 Implementation Forms

As previously noted, STEP separates the concept of data specification (data model) from implementation (data values). The Implementation Forms parts of STEP define standard formats for data instances and values. These formats have mappings between them and the EXPRESS language to enable checks for consistency and correctness. STEP currently only supports two implementation forms although more has been designed:

- The "physical file" format (published as ISO 10303-21, a.k.a. STEP Part 21) is an ASCII encoding form for data that conforms to any EXPRESS schema. There is nothing specified in the standard about how these files are to be exchanged.
- The Standard Data Access Interface (SDAI, published as ISO 10303-22, a.k.a. STEP Part 22) specifies standard functions that allow access to data within a database or application. The database or application needs a special interface to communicate with. The SDAI functions allow programs to manipulate product data defined by an EXPRESS model within the database or application.

Example: A car data instance, stored using STEP Part 21

```
ISO-10303-21
FILE_SCHEMA((‘car’,’person’));
ENDSEC

DATA
#1 = CAR ( “Ford” , “Orion” , 1989 , #2 )
#2 = PERSON ( “Julian”, “Fowler”)
ENDSEC
END-ISO-10303-21
```

These implementation forms are the big problem with STEP; it has no support for Web and Internet usage:

- STEP part 21 only defines how the files should look; nothing is said about how they should be distributed.
- STEP part 22 only specifies an API so that a database containing EXPRESS model data can be manipulated.

3.5.2 STEP or XML?

STEP and XML have been developed for much the same reasons, to describe and transfer data in a structured and standardized way, independent of the applications that are going to use the data. Therefore they have many similarities:

- Both are used to transfer data between applications.
- Both separate the data models from the actual data, in STEP through an EXPRESS model and in XML through a DTD.
- Both has support for checking the data instances against predefined models, as above.
- Both support modular development.
- Both supports direct interaction with databases through special interfaces.

With so many similarities what are the key differences that separates the two standards?

- **Tools**: STEP has been around much longer and has more tools developed to support it. But new tools are being developed for XML every day.
- **Type Checking**: STEP incorporates type checking, something that XML does not do (but an application reading the XML document could do it).
- **Data Models**: STEP uses the EXPRESS language to define models. The closest thing XML has is the DTD and it does not have nearly the same functionality as EXPRESS concerning constraints, heritage, etc.

- **Predefined models**: STEP has predefined generic models that are used to define the specific data models. No XML DTDs or namespaces have yet been developed to specifically support product data, but that does not mean that it can't be done.
- **Internet support**: XML documents can easily be distributed over the Internet using HTTP, there is no such support in STEP.
- **View**: XML has a stylesheet language that can be used to view the data directly.
- **Data Manipulation**: XML has a proposed query language that could be used to manipulate the product data stored in the XML documents.
- **Links to other documents**: XML supports in-line links to other documents so that not all the product data need to be stored in the same document. In STEP all data needs to be in the same document to be able to verify that the data conforms to the EXPRESS model.

3.5.2.1 Conclusion

What it all boils down to is this. XML isn't nearly a complete replacement of STEP; it does not have support for inheritance and constraints for the data models in the same way as EXPRESS has. XML could however be used for much the same goals and provide other functionality that STEP does not have, such as Web support.

3.5.3 STEP and XML?

One really interesting area, that now is being researched, is the possibility of combining the STEP and XML (or SGML) standards. This would give the developer the best of two technologies:

- STEP's powerful EXPRESS schema language, that offers a well-developed constraint formalism for defining product data.
- XML's support for the Web could greatly facilitate the interchange of STEP-conforming product information. STEP data could, of course, be interchanged using XML, given an appropriate XML-based information architecture that is widely understood and accepted.

There are currently two views of how this could be done.

One group wants to fully incorporate XML as an interchange format between STEP-compliant databases. To be used in this way would require a standard means of expressing the entire data model of a STEP-compliant database, together with a subset of its contents, in the form of valid (or at least well-formed) XML. The interchange package would presumably need to include appropriate DTDs (if any), Link files, and a robust universal addressing mechanism. The inclusion of stylesheets would presumably be optional, depending on whether, in addition to data interchange, the contents of the package were intended for display.

The other group does not think that XML is up to the task of fully interacting with STEP. This is because they think it is wildly unlikely that generic XML browsers can usefully support interactions with STEP source data. There are far too many specialized semantics. And if you can't usefully interact with STEP source code using a generic XML browser, what's the advantage of using XML to interchange STEP data? They instead suggest using SGML/HyTime [44] for interacting with STEP source data. XML should be used separately to providing access to STEP data on the Web.

It is not clear which of these views that is "correct". They both agree that there definitely is a place for XML in STEP, but they disagree about how it should be used. W3C hasn't yet made any comments on the matter. In the words of Jon Bosak, Chair of the W3C XML Group:
"I don't think that the burden of proof falls on the W3C group to prove that XML can meet the needs of STEP. I think that it's up to those who believe that XML is incapable of meeting those needs to tell us where it falls short."

3.5.4 STEP or SOX?

After I made the above observations, the SOX specification has been submitted to the W3C for review. This standard seems to be able to take over the responsibilities from STEP since it, unlike the XML 1.0 specification, **has** support for type checking, inheritance, and constraints.

Since it was submitted very recently, little or no feedback has reached the Internet community concerning it as a possible replacement of STEP. It is known that Microsoft has announced that they will support XML schemas in IE5, so at least the specification has some support.

Part 4: Trying out XML

In theory XML looks really promising, but as all Internet programmers know - theory is one thing and reality is often another. It is therefore necessary to try the specification out in a real project to find the advantages and disadvantages compared to earlier techniques and specifications.

"Given that XML was designed for use over the Web (right?) and it has been in gestation for 2 years I find it incredible that XML has not done anything useful in public view. Lots of hype in the magazines, etc. but nothing tangible to show for it. Tangible in the sense that I can show a non-XML person something that will interest them.

XML was effectively launched in spring 1997 at WWW6, Santa Clara. It's 15 months since then and over a year since the first draft of the XLink spec was released. And as far as I can see there are virtually no useful applications that have been created. I now find it difficult to convince people that XML is useful, other than by repeatedly stating it as an act of faith."
--Peter Murray-Rust on the xml-dev mailing list: September 16, 1998.

There has been an enormous amount of hype surrounding XML since the first working draft was released back in November 1997. But it is only recently that some members of the Internet community actually have started doing something with the standard. When Decerno decided to try out XML in a real project (and not just in a test environment) they put themselves in the absolute forefront of XML developing.

4.1 The application – Pastill

In October 1998 Decerno received an order from Omsorgsnämden in Sweden for an Interned based application that would be used to view and gather statistics from Stockholm's many "Psykisk Barn och Ungdomsvård" (PBU) receptions. The name for this application was decided to be **Pastill**. The Pastill project was something of a pioneer project for Decerno and it was decided that we would use it to evaluate and try out several new technologies and methodologies, including XML.

In the following chapters I will first describe the Pastill application – what it does, how it does it and where XML is used. After that I will comment about how well XML worked out in Pastill – was it as good in practice as it looked in theory? Finally, I will make some comments about my experiences concerning Pastill as a project – time planning, working in a team, satisfying customer demands, and so on.

4.1.1 What should Pastill do?

Pastill is a Web-based system for gathering patient- and time-statistics. There are two types of demands put on the application, functionality demands and resources/technology demands.

4.1.1.1 Functionality

The functionality that Omsorgsnämden wanted Pastill to have was to over the Web (Intranet) enable the user to view, change, input and/or search data concerning:

- Personal data about patients (i.e., date of birth, country of origin, relatives, and so on).
- Data concerning patient visits and the time spent on these visits by the staff at the different sites.
- Other types of statistics (for instance, the average time from the first visit to the last visit for all patients).

4.1.1.2 Other requirements

When Decerno received the order for Pastill, there were a number of requirements that the system had to fulfill besides having the wanted functionality.

- **Low bandwidth usage**: Since the Intranet at Omsorgsnämden for the most part consists of dial up modem connections, Pastill couldn't be allowed to use to much bandwidth because of the limited bandwidth available on these connections.
- **Work on Internet Explorer 4.0 Service Pack 1**: The client part of the system had to be able to work on the Internet Explorer 4.0 Service Pack 1 (IE4.0 SP1) platform from Microsoft. (This demand automatically disqualified Pastill from making use of the built in support for XML that IE5.0 has).

- **Automated installation on the clients**: The installation of the client components should be automated so that the system does not require a hands-on installation of each client.

4.1.2 Design

With these requirements in mind we (the team of three people behind Pastill) started the development process by looking at a suitable design. As all programmers know, the design is one of the most important parts (if not THE most important part) of any development. If the design is wrong it does not matter how good you make the implementation, it simply will not work.

In Pastill we wanted to do it right the first time and go for a scalable three-tier Web design right from the beginning. At this stage I also planned to try out two different technologies for the Client-Server communication but as I will return to later, a broken time-table put a stop to that idea and I only made an XML implementation.

However, an overview of the design can be seen in figure 13, each part will be more thoroughly described in the following chapters. Since many of the technologies used in Pastill are beyond the scope of this document, I will only give the reader a short description of these technologies in order to give a perspicuous view of the application. The goal is to allow the reader to understand how these technologies have been used in Pastill together with XML to make it all work together. The interested reader is referred to the documents given in the bibliography [20,31,40,41,42,49,50,56].

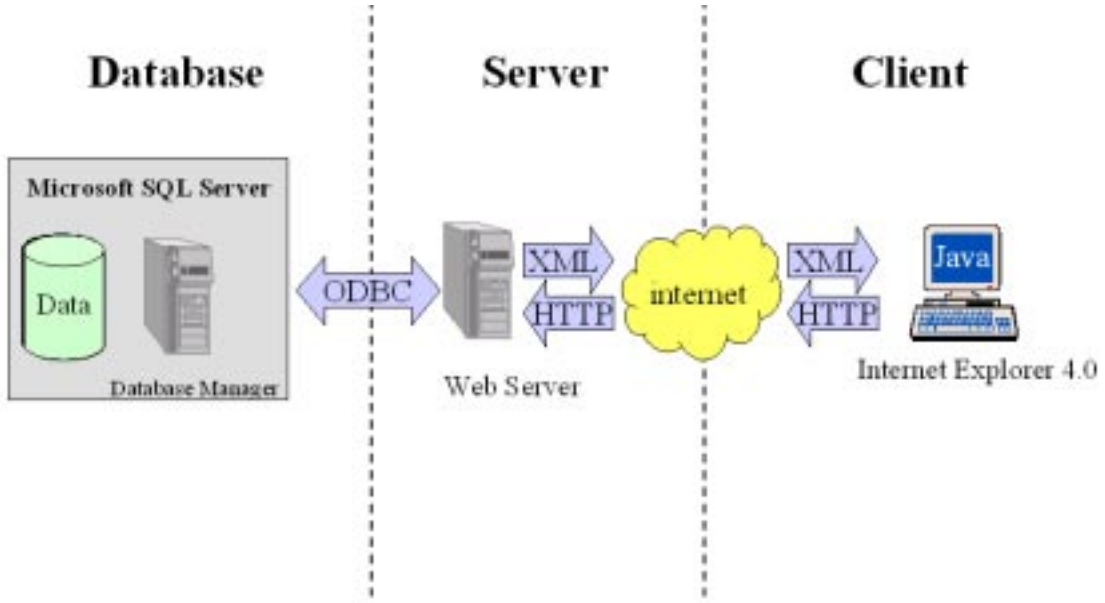


Figure 13: Overview of the three-tier design of Pastill

4.1.2.1 Database

The database in Pastill is a standard Microsoft SQL Server v6.5. It consists of 42 tables with a total of about 170 columns. It supports an Open Database Connectivity (ODBC) [49] interface that the Server uses to access the database. The Server uses Microsoft ActiveX Database Objects (ADO) [31] to interact with this ODBC interface to store and retrieve information from the database.

Open Database Connectivity (ODBC) is a Microsoft established standard [49] that enables software developers to create applications that can work with a number of SQL-based data sources. With ODBC, applications needing data from a data source can generate ODBC calls, and these calls are then translated by an ODBC driver into the native SQL of the data source being accessed.

ActiveX Data Objects (ADO) is a Microsoft established technology [31] that can be used by developers to add database access to their applications. It's capable of condensing otherwise complex and lengthy programming tasks into simple-to-use statements. Using ADO you can connect to databases, issue queries, and gather responses. ADO lets you step through the records you have queried for, make changes, and send those changes back to the database. ADO does this without requiring the programmer to know the intricate details of communicating directly with the database to perform the task set before him.

4.1.2.2 Server

The server in Pastill consists of two parts – a Web Server and a Microsoft Transaction Server (MTS) [42]. The Web Server handles all communication with the client and the Transaction Server handles most of the communication with the database.

The Transaction Server contains Microsoft Java Component Object Model (COM) objects that are used to store data in the database. These objects are stored in Transaction Server in order to speed up the application, and to give the advantages of a component-based solution. The speed-up is achieved through Transaction Servers ability to reuse components after their release, i.e., the objects only need to be created once and thus it takes shorter time to reuse them.

The Microsoft Component Object Model (COM) [40] is a way for software components to communicate with each other. It's a binary and network standard that allows any two components to communicate regardless of what machine they're running on (as long as the machines are connected), what Operating System (OS) the machines are running (as long as it supports COM), and what language the components are written in.

The Web Server is used to store and process Microsoft Active Server Pages (ASP) pages. These ASP pages make up the framework for the entire application, both on the server and on the client. Each of them is used to hold the different components together and to pass data between the components. It is important to understand that the ASP pages are NOT used to pass data between the client and the server, where XML is used. The ASP pages are however used to create XML documents as will be later described.

Active Server Pages (ASP) technology is a powerful Microsoft Web Server feature [50]. In brief, ASP allows developers to mix HTML, server- and client-side scripts in a single file. Thus allowing developers to control both how the server interacts with the client (server-side scripting), and how the user interacts with the client (client-side scripting).

The Web Server also has a Java COM that handles all the XML traffic on the server. This object and the corresponding object on the client will be described below.

4.1.2.3 Client

The client consists of two parts, a GUI and a Java Applet that corresponds to the Java COM object on the server that I just mentioned. It is held together in the same way as on the server using ASP pages.

The GUI on the client is programmed using Visual Basic 6.0. In the beginning we planned to use Microsoft Windows Foundation Classes (WFC) [56] to build the GUI. Early tests however showed that in order to make this work the client would have to download huge runtime libraries, and that was not desirable due to the limited available bandwidth.

4.1.2.4 Where does XML fit?

Pastill uses the XML v1.0 Specification to define XML documents that are transferred between the client and the server. Both the client and the server have Java objects running that sends and receives XML documents according to the user's actions.

The client Java object has two functions; it is used to:

- receive information (XML documents) from the server and to process that information so that it can be presented to the user.
- collect information from the GUI that it sends (using XML documents) to be saved by the server in the Database.

The server Java COM object has the corresponding two functions, it is used to:

- send information stored in the database to the client (using XML documents).
- receive information (XML documents) from the client that it should save in the database.

4.1.3 How Pastill works - Flowcharts

It is always very hard to try and describe the underlying structure and functionality of an application to someone who hasn't been involved in developing it. But in this chapter I will try just that to give an overview of Pastill, and to try and show just how Pastill uses XML.

The easiest way to understand how any system works (I have found) is often to view a chart that shows the different transactions that can occur in the system. By transactions I mean how different actions by the user cause different flows of events and data to stream through the application.

In Pastill, all transactions are initiated by a user action. Depending on the type of the action, two different but similar types of transactions are initiated. The type of the resulting transaction depends on if the user action causes data to flow **from** the client or **to** the client. In this chapter we will examine both of these two transactions from two different viewpoints – that of the client's and the server's.

4.1.3.1 The Client Requests Data

This type of transaction was initiated when the client requested some data stored in the database. First I will look at how this transaction looks from the client's viewpoint and then how it looks from the server's. I will begin with a graphical view and then explain how and in which order things happen in the application.

4.1.3.1.1 Client Transaction

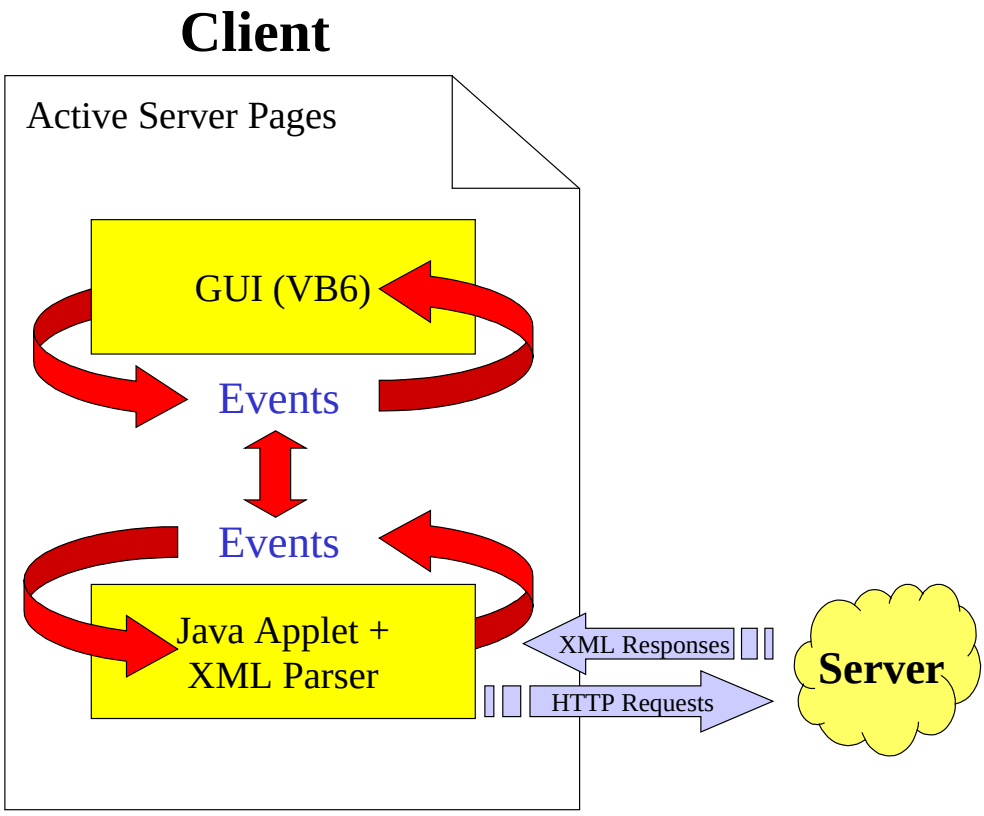


Figure 14: Client Transaction (data is passed from the server to the client)

1. The user initiates the transaction by pressing a button in the Visual Basic GUI. For instance, the user wants to view a list of all patient journals from the Lidingö PBU-Reception.
2. This results in an event that is sent from the GUI, to the ASP page that the GUI lies in.
3. The ASP page receives the event and, depending on the event type, runs a method inside a Java Applet that lies waiting in another ASP page on the client. This Applet can be thought of as the "engine" of Pastill.
4. This is where things get interesting. The method in the Applet issues an HTTP request for an ASP page located on the Web Server. In the HTTP request header, the Applet includes action specific parameters (for instance, the desired journal number) that are received by the requested ASP page.

5. Exactly what the Server does will be shown later in the next section, but the result is that an XML document is sent back to the client with the requested data.
6. The Applet receives the XML document and feeds it to an internal Java XML parser from IBM AlphaWorks named “XML Parser for Java” (XML4J) v1.1.9 [57] that will be described later. The XML4J parser parses and validates the XML document. It then constructs Java classes that are passed on to the Applet to read and manipulate the received XML document. The Applet uses these classes to process the XML document.
7. When the Applet has finished processing all the data stored in the XML document, the gathered data is sent in a number of events to the ASP page.
8. The ASP page receives the event and passes the data along to the GUI.
9. The GUI receives the information and presents it to the user.

4.1.3.1.2 Server Transaction

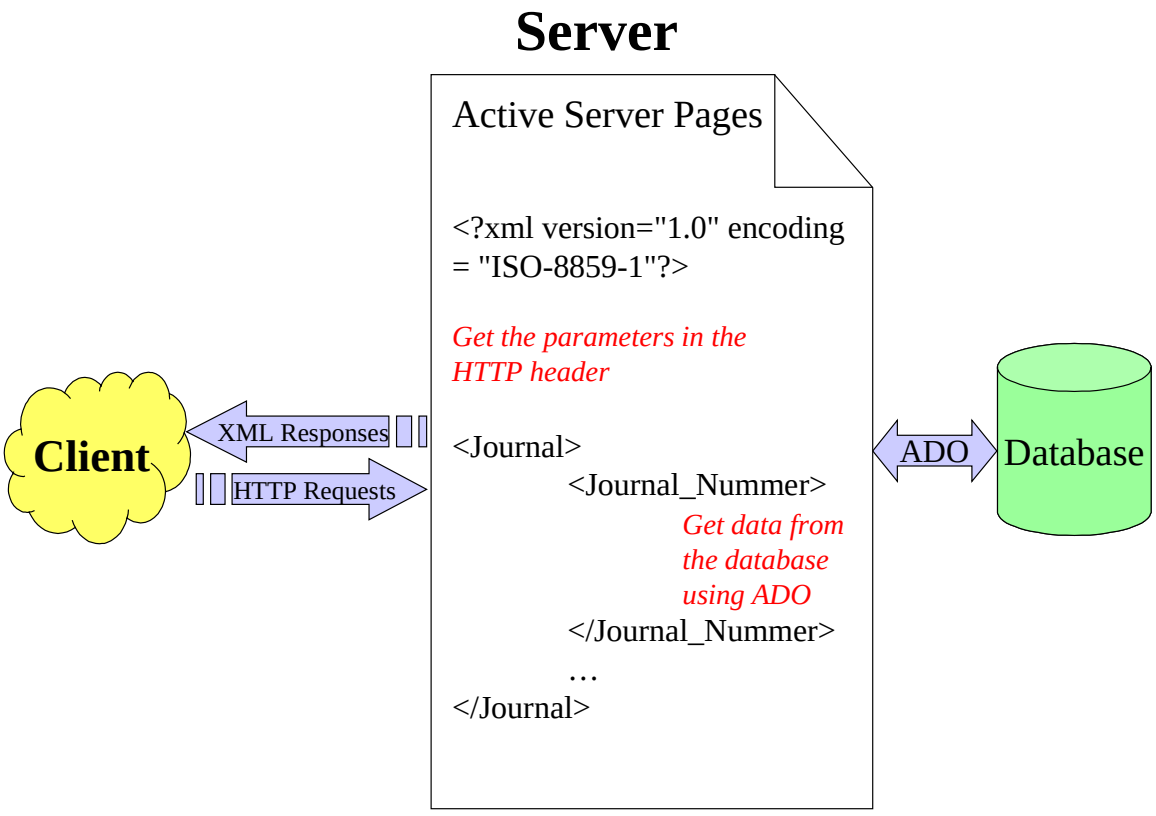


Figure 15: Server Transaction (data is passed from the server to the client)

1. The client initiates the transaction by sending an HTTP request for an ASP page located on the Web Server.
2. The Web Server loads the requested ASP page and prepares to execute the server-side scripts (marked in italics, in figure 15). The loaded ASP page almost looks like an XML document. But instead of having values inside of the XML leaf elements, it has server-side script-code, henceforth referred to as leaf-scripts.
3. The first thing the server-side script does, is to retrieve the parameters that where passed in the HTTP header.
4. Then, in every tag, the leaf-scripts are run to fetch the correct values for this element from the database depending on the input parameters.
5. The leaf-scripts uses Microsoft ADO/ODBC to retrieve the data corresponding to the leaf tag names from the database. For instance, the value inside `<Journal_Number> </Journal_Number>` is fetched from the `journal_number` column in the journal table from the database.
6. The data is returned and all the leaf-scripts are replaced by their fetched data.

❼ When all the leaf-scripts have been processed, the original ASP page has been transformed into a XML document containing the requested data!

❽ The ASP/XML document is sent back to the server as a response to the HTTP request.

Connect the described client and server transactions and that is basically how all data gets transferred from the database to the server.

4.1.3.2 The Client wants to store data

The user initiates the second type of transactions in Pastill when he/she wants to store data in the database. As before I will first show how this transaction looks from the client's viewpoint and then how it looks from the server's.

4.1.3.2.1 Client Transaction

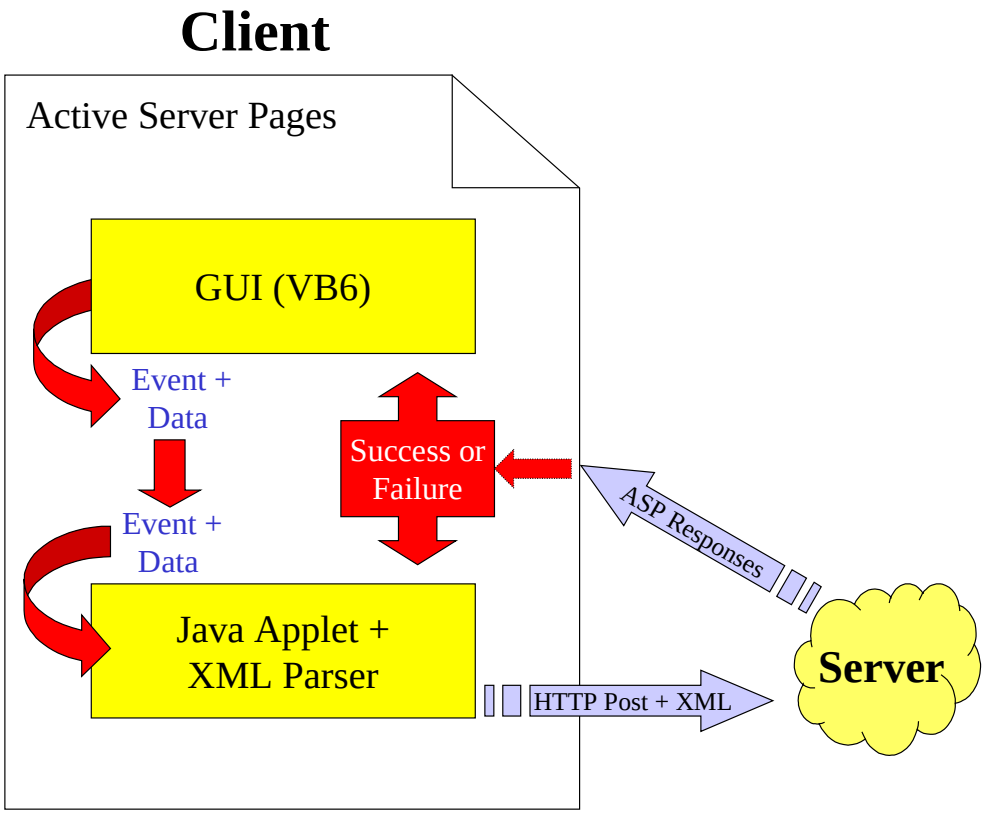


Figure 16: Client Transaction (data is passed from the client to the server)

❶ The user initiates the transaction by wanting to save some data in the database. For instance, the user wants to save a new journal for a new patient.

❷ As before this results in an event that is sent from the GUI - to the ASP page that the GUI lies in. But this time the event contains data that the user wants to save.

❸ The ASP page receives the event and, depending on the event type, runs a method inside the same Java Applet as before.

❹ The Applet retrieves the data that was sent in the event and passes it to the internal XML4J parser.

❺ This XML parser creates a new XML document based on that data, and returns it to the Applet.

❻ The Applet takes the returned XML document and posts it to the server using an HTTP Post request.

❼ I will in the next section show exactly what the server does with this HTTP Post request. But it always results in this: The server returns an ASP page that tells the client-side ASP page, if the data has been saved correctly or if some errors occurred.

❽ The client-side ASP page passes on the received success/failure to both the GUI (to be presented to the user) and to the Applet (because it needs to be up-to-date).

❾ The GUI receives the success/failure and informs the user accordingly.

4.1.3.2.2 Server Transaction

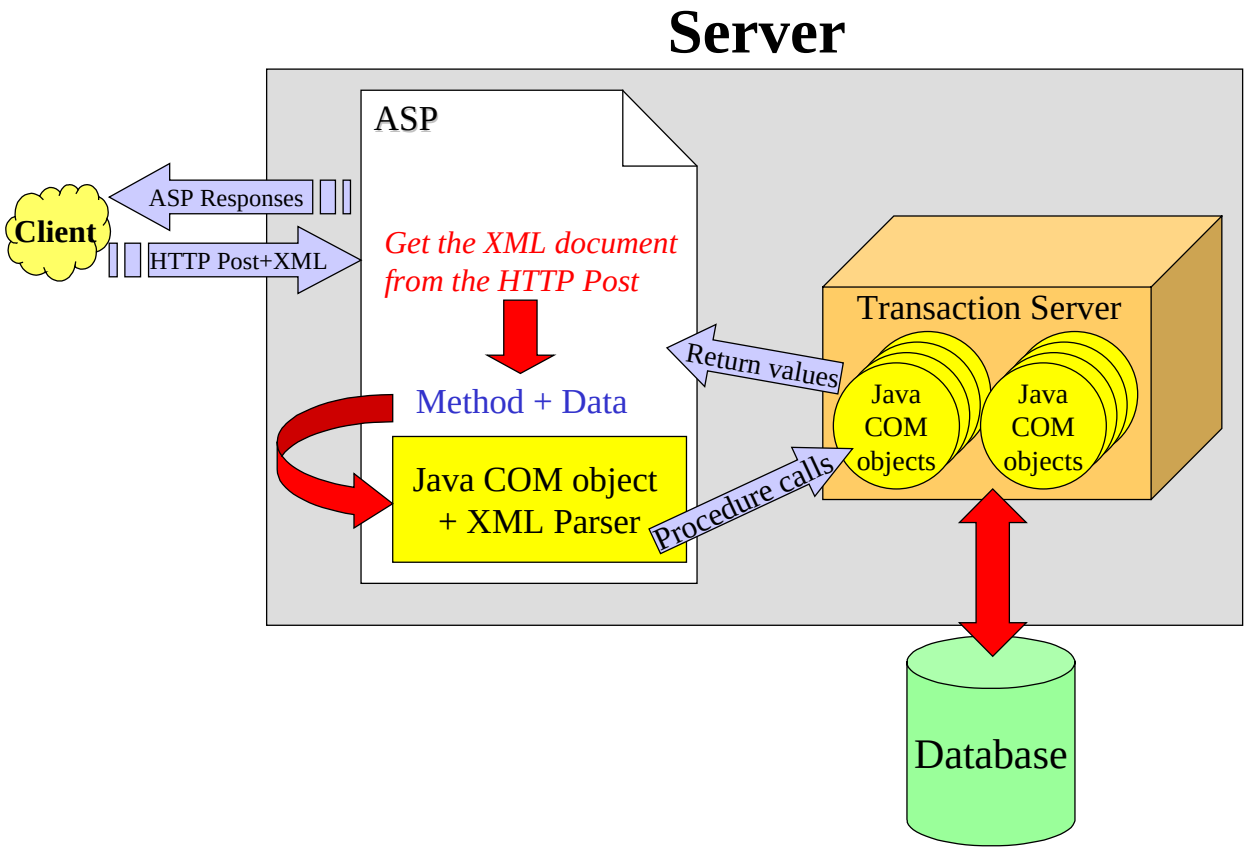


Figure 17: Server Transaction (data is passed from the client to the server)

❶ The client initiates the transaction by sending an HTTP Post request to an ASP page located on the Web Server. The HTTP post contains an XML document embedded in the request.
The ASP page is loaded on the server and the server-side scripts are initiated.

❸ The first thing that these scripts do is to retrieve the XML document from the HTTP post.

❹ Then a method is called in a Java COM object that lies embedded in the ASP page, the retrieved XML document is sent as an argument to the method.

❺ The method in the COM object takes the XML document that it received and passes it to a server-side, internal, XML4J parser. The XML4J parser parses and validates the document, it then returns Java classes to the COM object to read and manipulate the information.

❻ The COM object uses the returned classes as it calls procedures inside other Java COM objects. These objects are located within the Microsoft Transaction Server (MTS). The sole purpose of these objects is to store information in the database. They are highly specialized and each is used to store one little piece of the data. They are located within MTS because MTS has automatic transaction handling (including rollback if something goes wrong) and connection pooling.

❼ Each object in the MTS takes its little piece of data and tries to store it in the database. If all MTS objects succeed in saving the data in the database, the ASP page is notified of their success. If one or more of the MTS objects fail, nothing is saved in the database i.e., rollback occurs, and the ASP page is notified of their failure.

❽ The ASP page gets the MTS objects success/failure return value. The ASP page reformats itself according to the return value and is returned to the client to inform it of if the transaction succeeded or failed.

4.1.4 At the Heart of Pastill – an XML Parser

A vital part to make Pastill work is the Java XML parser that is used both on the client and on the server. As can be seen above, it is used once every time the user wants to retrieve data and twice when the user wants to save data.

As I have mentioned in an earlier chapter, there are today a number of Java XML parsers available from different sources. One of the decisions I had to make early in the design process was to choose which XML parser that we would use in Pastill. I had five requirements that I wanted the parser to fulfill:

- It should not be a beta (i.e., it should not contain too many bugs).
- It should have backing from a major company.
- It should be reasonably fast.
- It should have a license agreement that won't require Decerno to pay a lot of money to be able to use it.
- And most important, it should provide the functionality that we needed (i.e., both the ability to parse AND the ability to create XML documents).

I had some previous experience of an early beta of the X4J parser from Microsoft after trying it out in several small test applications. But I wasn't quite happy with the functionality it provided, also I perceived it as too slow to use in a real application.

The parser I finally choose after reading about several others, was the IBM AlphaWorks XML Parser for Java (XML4J) version 1.1.9 [57]. This parser is a validating XML parser written in 100% pure Java. The XML4J package contains classes and methods for parsing, generating, manipulating, and validating XML documents. XML4J is believed to be the most robust XML processor currently available and conforms most closely to the XML 1.0 Recommendation. XML4J also supports the DOM Level 1 Specification, the Namespaces in XML Specification, and the unofficial (not a W3C Specification) SAX Specification.

SAX [36] stands for Simple API for XML and it is an extremely popular (almost ubiquitous) event-based interface for any XML parser. SAX is somewhat of a grass-roots effort, spearheaded by David Megginson, of the xml-dev mailing list. SAX is the basis for quite a few Java APIs. The idea behind SAX is to provide an interface by which any Java application can access any XML parser, provided the parser has a SAX driver. Virtually every major XML parser either supports the SAX interface directly or indirectly via third-party drivers. SAX uses an event-driven model: the parser identifies an element, resulting in that element's event handler being invoked. It is optimized for applet/browser use and can plug-in to any particular parser.

The reason for choosing the XML4J parser was that it fulfills all of my above requirements. Furthermore it has a well-written documentation that makes it much easier to use than the X4J parser I had previously worked with, plus all the source code is included with the parser. One disadvantage with this parser is however that the package is quite large (about 1.2 MB), but I considered all it's advantages to outweigh this disadvantage [58].

Pastill interacts with the XML4J parser through methods defined by the W3C DOM Specification that is implemented by the parser. This makes it (in theory) very easy to simply replace the XML4J parser with another parser that also implements the DOM Specification, without having to rewrite the entire application in the process.

4.1.5 Walkthrough of Pastill

In this chapter I will give a very short overview of what the user will see when using Pastill. I will also very briefly mention what techniques have been used. The whole client part of the application runs on Internet Explorer 4.0 SP1, which was also one of the requirements made of the system.

4.1.5.1 Layout

When we designing the GUI in Pastill we wanted all views to be as consistent as possible, hopefully this shows in the application so that the user will not feel lost. The layout is displayed in figure 18. On the left is a menubar that is always visible, it shows what options are current available for the user. The rest of the display is made up of the workspace where the user interacts with the application by viewing or adding data.

In a previous chapter I stated that Pastill was held together by a framework of ASP pages, in figure 18 can be seen just what I meant by that. More or less all the views in Pastill consist of one or more ASP pages with Visual Basic components embedded in them. The menubar on the left in figure 18 is a Visual Basic version 6 (VB6) component; ASP makes up the rest of the view. The Java objects are never visible to the user, but are hard at work behind the scenes.

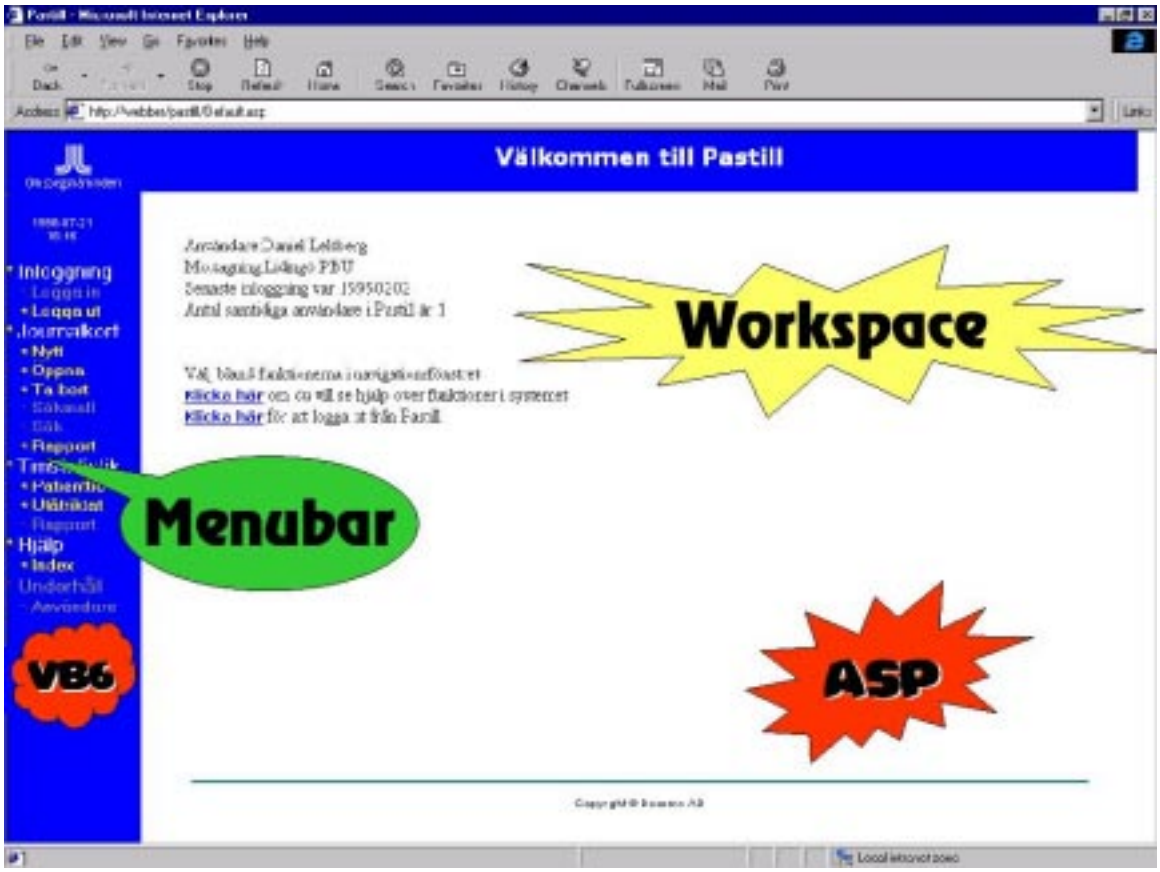


Figure 18: The layout in Pastill

Since this layout never changes I will save some space in the following chapters by only showing the different functions that exists in Pastill and not show the entire layout every time. All the listed components are VB6 components that are displayed on the workspace.

4.1.5.2 Entering Pastill

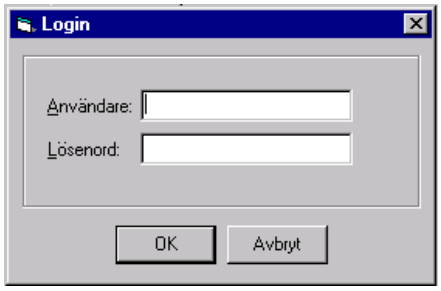


Figure 19: Logging in to Pastill.

The first view that the user is confronted with is the login page. The user is required to input a username and a password. This is compared with the username and password stored in the database. Since there is so little data do transfer this is all done in ASP scripts, completely without the use of XML or Java.

4.1.5.3 Retrieving Journals

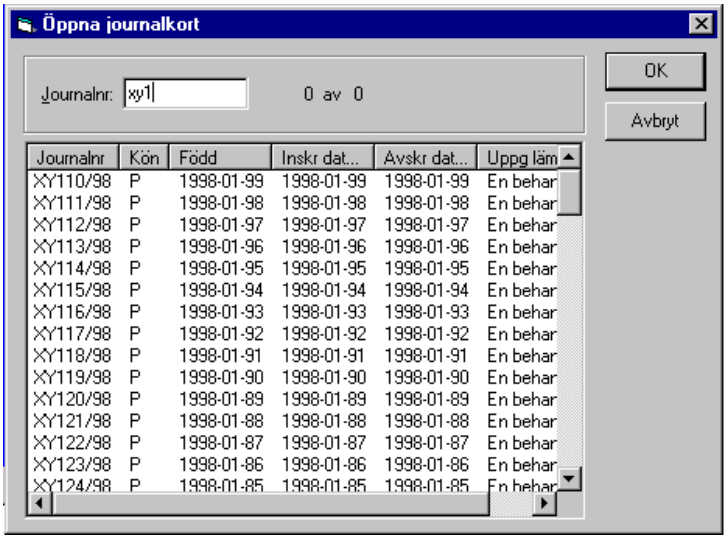


Figure 20: Retrieving a list of journals.

This function allows the user to retrieve lists of journals that meet certain criteria. The list is fetched using XML and the technique described in section - **4.1.3.1 The Client Requests Data**.

4.1.5.4 Working with Journals

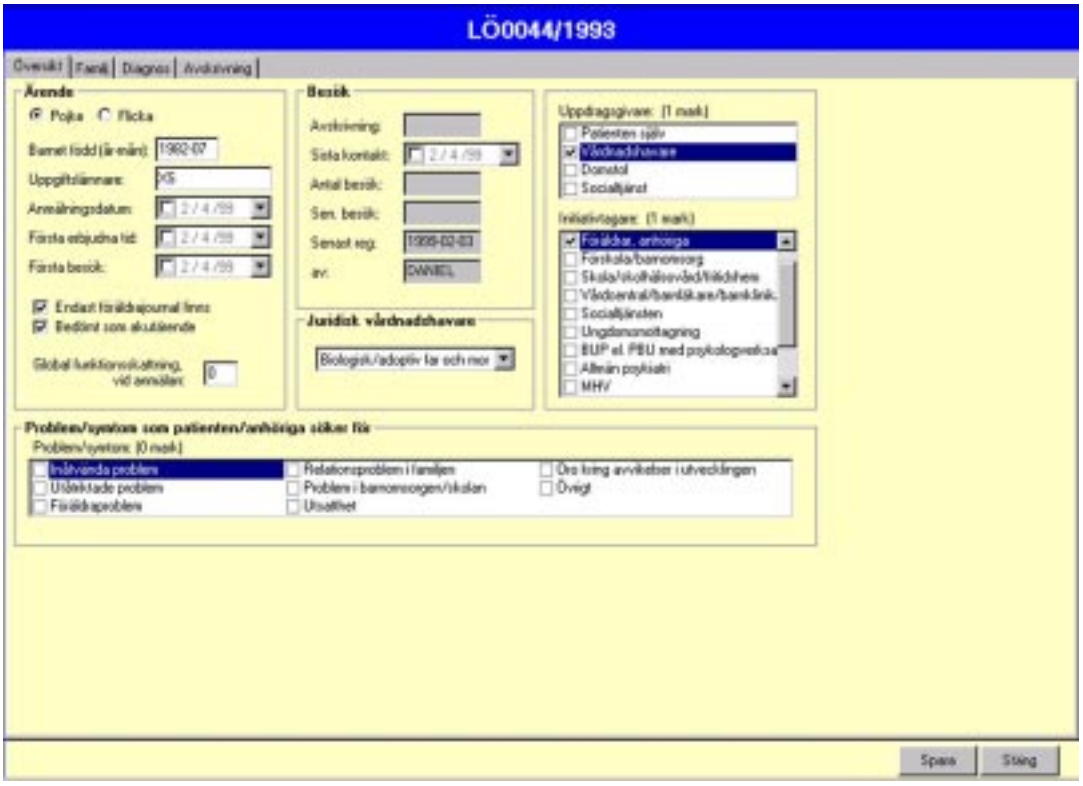


Figure 21: Working with journals

This is the view used to work with journals. The user can view existing journals, change information in existing journals, or create new journals. When the user wants to fetch a journal to view or edit, it's done using XML and the technique described in section - **4.1.3.1 The Client Requests Data**. When the user has made some changes to a journal or created a new journal and wants to save that information, it's also done using XML, but this time using the technique described in section **- 4.1.3.2 The Client wants to store data**.

4.1.5.5 Working with patient visits

Figure 22: Working with patient visits.

This view is used to input and view data concerning patient visits. Behind the scenes it works exactly like the **Working with Journals** view.

Basically these two examples present a sample of what kind of views that exists in Pastill. There are some more views, for example the Report view where the user can see different reports about statistics gathered by the system, but showing these do not contribute very much to the understanding of what Pastill is used for.

4.1.6 Experiences learned by using XML in Pastill

By reading the **Design** and **Overview** chapters you should now at least have a basic understanding of how XML was used in Pastill. In this chapter I will present the advantages and disadvantages that I have encountered when using XML in Pastill. The statements are made from a very practical point of view, based on my experiences while programming and writing the XML documents for Pastill.

As I have mentioned before, and will mention again in the chapters Future Work and Time Planning, I didn't have time to make a second implementation of Pastill with a technology other than XML. So unfortunately I don't really have something to compare this implementation with. Some of the statements made are therefore based on my, and my colleague's, previous experiences of other technologies when comparing them to XML.

4.1.6.1 Advantages of using XML

What specific advantages of using XML did I encounter? I will not list the abstract advantages such as that XML enables easy human reading of the data, or that in the future it will be very easy to do a "look-only" version of Pastill using XML and XSL or CSS. Instead I will list the advantages that we noticed when using XML in Pastill.

4.1.6.1.1 Very easy communication

It is very easy to write the code that handles the communication when using XML, basically because such code is not needed. All that that is needed to communicate is to use HTTP requests to transfer XML documents, and this do not require much code in order to work. All other available technologies (Java Sockets, DCOM, CORBA, and so on) need a lot of code that handles the communication in order to work. In Pastill, the entire XML communication takes up about 30 lines of code.

4.1.6.1.2 Bandwidth Usage

One of the requirements made on Pastill was that it shouldn't need much bandwidth to achieve good performance. Using XML we have fulfilled that requirement nicely. Almost all data in Pastill is transferred between the client and the server using XML and HTTP. We have tested Pastill on a number of networks and discovered that the waiting time is almost independent of the network quality. Instead it is the database accesses and in a lesser degree the client processing that make up the waiting time.

4.1.6.1.3 No need to have a real-time connection with the Server

In the beginning of this paper I listed some questions that this thesis should answer. One question was if XML in any way enabled an alternative to having a real-time connection with the server by for instance, demand-driven delayed communication with the server? This means holding off any client – server communication until a connection can be established and meanwhile caching all transactions on the client.

When working with Pastill, I have discovered that the answer to that question is yes, XML does not need a real-time connection. All the important transactions in Pastill are all document (file-) based. Instead of sending the documents to the server as soon as they are created on the client, there is no problem holding off the transaction if the client should discover that the connection for some reason is down. This functionality isn't currently incorporated in Pastill, put it would be quite easy to implement it.

4.1.6.2 Problems with using XML

I did encounter some problems with using XML. Mostly because the available tools still need to mature, but also because I wasn't used to handling these types of data structures. I have discovered that making good XML structures require a lot of practice and hands-on experience, otherwise you just don't see where the problems can occur.

4.1.6.2.1 It's hard to write a good structure

It is hard to come up with a good structure for the XML documents right away. I have had to revise the structure of the XML documents in Pastill a number of times, and each time the XML documents change I have to change the code that process them to reflect the changes.

It is especially hard if you want a structure that enables you to program general element handling for many similar elements at the same time. I had to solve this by insert dummy elements as children of the elements I really wanted to process. These dummy elements all have the same tag names so I can recognize them as belonging to a group of elements that should be processed in the same way.

Example

```
<Journal>
        <Journal_Number>
                <Text> 23 </Text>
        </Journal_Number>
        <Sex>
                <Text> Male </Text>
        </Sex>
        <Family>
                <Alternative> Real Mother </Alternative>
        </Family>
        <Treatments>
                <Alternative> Child therapy </Alternative>
        </Treatments>
</Journal>
```

Using the above technique I can recognize all `<Text>` and `<Alternative>` dummy tags and treat these elements the same. If I wouldn't use the dummy tags it would be easier to read the XML document, but when programming the code I would have to recognize all the different tag names that should be treated the same. In this example I want to treat `<Journal_Number>` in the same way as `<Sex>`.

I also encountered the previously described attribute - element problem. When should I use an element and when should I use an attribute? I have not found a simple answer to the question but I can say that programmatically I have found it easier to use elements instead of attributes, especially if you have more than one attribute.

4.1.6.2.3 Tags take up lots of space

When looking at an XML document, it is easy to see that the tags themselves make up much more of the document than the actual data that is stored. This stems from the fact that the XML 1.0 specification encourages the writer to write out full tag names and not use abbreviations. This makes the document easier to understand for humans, but it also has as the effect that the amount of "useful" data stored in the document compared to the document size is surprisingly small. So the prize for readability is a very high amount of overhead in the XML documents.

This is not a real problem in Pastill since all the generated XML documents are quite small, but when handling large quantities of information I think this is definitely something to have in mind. However, Lempel, Ziff, Welch (LZW) compression [59] would be well suited to reduce the size of large XML documents because LZW compression is dictionary based. Once a string is in the LZW dictionary the cost of repeating this string in the document is very small, note that this is the compression used by Adobe's Portable Document Format (PDF) standard along with many other applications.

4.1.6.2.4 Too much string handling slows down the application

Any XML document by definition contains textual information; it's the whole idea. Processing text is unfortunately something that generally most programming languages are bad at. They have much better performance when handling for instance integers.

This is a well-known problem and the XML parsers are optimized as much as possible to speed up the processing. The real problems occur if you programmatically need to alter the XML documents before sending them to the parser. Then you can have serious performance problems, even with relatively small XML documents.

We had this problem during the development of Pastill, but not in the release version. This stemmed from a combination of two things.

- We used Microsoft Visual Studio v6.0 (VS6) to develop Pastill. In order to debug Web pages, VS6 puts debug-information in a `<!-- -->` comment first in the ASP pages that creates the XML documents.
- The IBM AlphaWorks XML Parser needs the XML declaration first in the XML documents; otherwise it will refuse to parse the document.

The result was that we couldn't run debugging on the ASP pages because this caused the XML parser to stop working. In order to get around this problem so that we could debug, I put in a little String handling that removed the comments before feeding the XML document to the parser. This considerably slowed down the application.

Again, this is not a problem in the release version of Pastill since no debugging is used there. We simply feed the XML documents to the XML parser directly and we are quite happy with the performance we get out of the application.

4.1.2.6.5 Problems with the XML4J Parser - Bugs

As is still the cased with almost all available XML tools, I discovered that the XML4J Parser has some bugs that were quite hard to detect.

- The XML4J Parser does not allow comments `<!-- -->` or blank-rows in the beginning of an XML document. The string `<?xml version="1.0"?>` has to be the absolutely first thing that the XML document contains. If it is not first, the parser throws a Java exception and refuses to parse the rest of the document.
- The XML4J Parser goes into an infinite loop when the parsed XML document has one or more Swedish letters (åääö) in the XML comments (I have not dared to even try and use Swedish letters in the tag names).
- The XML4J parser does not allow nested elements on the same row. For instance, the XML fragment

  ```
  <Journal> <Journal_Number> 4 </Journal_Number> </Journal>
  ```

  results in the Parser throwing a Java exception and refusing to parse the rest of the document, but the fragment below works just fine.

  ```
  <Journal>
        <Journal_Number> 4 </Journal_Number>
  </Journal>
  ```

4.1.3 Pastill as a project

Pastill was the first real project that I have been involved with a real customer and strict deadlines, and naturally I learned much from it. There have been several books written about project orientated software development and I will not attempt to go into any deep analyses of the subject. I will simply present my experiences from the Pastill project.

4.1.3.1 Working in a team

A team of three people was involved in developing Pastill:

- Daniel Lekberg was the leader of the project. He also developed the VB6 GUI, most of the ASP pages and the database design.
- Leif Pettersson was involved in creating the MTS objects and their connection to the database.
- Me myself (Pontus Norman) developed the XML processing Java components and the XML document structure.

The problem with team development I have found is that there can be a lack of communication between the team members. This was a problem when developing Pastill since all the components are so tightly integrated. We have had some problems making all the components work together for this reason. If one person makes some changes, these changes often directly affect the behavior of another component that is out of that person's control.

4.1.3.2 Time planning

Originally I planned to do two versions of Pastill: one that used XML for communication, and another that used some other standard in order to be able to compare the two. Unfortunately the first implementation has taken much more time than anticipated so I haven't had the time to make that second implementation.

Time planning is one of the most difficult things in a project. It turned out that we were overly optimistic in the time planning of Pastill. The first specification for Pastill was written in July 1998. At that time we thought that the application would be finished by the end of December the same year. It is now February 1999, and the application still isn't quite finished. All the major work is done, but it still has some bugs that need to be fixed.

So why did the project take so much longer than anticipated?

The biggest reason is that the purchaser of Pastill (Omsorgsnämden) kept changing the specification during the development. This caused us to have to remodel the database a few times, and changes in the database reflect directly on the code in every level of the application.

The second biggest reason was that we used a large number of different technologies that needed to work together. As I stated before, Pastill was something of a test project used to try out new technologies, and as such it served it's purpose. It proved harder than expected to make the ASP, VB6, Java, and COM components to be installed correctly and play together and this was an important discovery to make. If we had to do Pastill all over again from the beginning, I would argue for a simpler solution with fewer technologies that had to work together - perhaps to write the whole application in Java and XML.

A third reason was that we soon discovered that we used a very unstable development environment. As I have previously mentioned we used Microsoft Visual Studio v6.0 (VS6) to develop almost all the parts in Pastill. VS6 is a good development tool as long as it works, but unfortunately this is seldom the case. We have had a number of problems with this tool and I have personally have had to reinstall it a number of times because it suddenly stopped being able to perform certain tasks.

4.1.4 Summary – Pastill

One of the purposes with this thesis was to find out if XML works in reality and not just in theory. The goal of my involvement in Pastill was to find out if the XML can be used as an alternative to previously existing client – server communication technologies.

Pastill is a commercial application developed for Omsorgsnämden in Sweden to view and gather statistics. Pastill uses the XML version 1.0 Specification to create XML documents that are passed between a Web-server and a Web-client. In order to process these XML documents, Pastill incorporates an XML parser from IBM AlphaWorks.

The overall impression that everyone involved with Pastill has of XML is a positive one. With Pastill we have proven to ourselves that XML already is mature enough to use in a commercial application. We found some problems, but they were never serious enough that we ever considered using a technology other than XML. We experienced real advantages such as easy development, fast communication, and low bandwidth usage.

The conclusion drawn from Pastill must be that XML not only works, but also works well, as a practical data communication standard.

4.2 Future work

In this thesis I have scraped at the surface of a dozen specifications and technologies, however there is a lot left to do.

4.2.1 New Specifications pop-up all the time

I have been following the XML development now for more than 6 months, throughout this time there have been a continuous flow of new specifications, based on XML v1.0 Specification, that have been sent to the W3C for review. To be in the front-line of XML development it is necessary to examine these new specifications as they are presented.

4.2.2 Continued follow-up as Specifications evolve

Most of the specifications described in this document still have a long way to go before becoming W3C Recommendations. It is plausible that at a lot of them will be changed along the specification process. It is therefore necessary to continue, and on a regular basis follow up how the standards evolve and what possible impacts this have on their future usage.

4.2.3 Continued follow-up of software companies' XML efforts

The software companies' XML efforts have only just begun. The current lack of tools holds back widespread usage of XML. More and more tools are becoming available and it is important to continue and monitor what tools are developed. Especially interesting will be to follow what Netscape and Microsoft do with their Internet browsers and what support the database suppliers will present for XML.

4.2.4 Continued follow-up of XML-EDI effort

One very interesting proposed usage of XML is as a replacement of EDI. One such effort is currently under way by the XML/EDI Group. If they succeed in developing a widespread and commonly accepted XML standard for EDI messages this will have a big impact on the Internet community. It is therefore necessary to follow-up on these efforts.

4.2.5 Continued follow-up of XML-STEP effort

Using XML as a replacement of STEP does not seem to have a widespread interest in the Internet community today. To the best of my knowledge, no organized effort is currently made to replace STEP with XML. Perhaps because the STEP standard is mostly used in specialized environments such as CAD/CAM development. However it wouldn't surprise me if such an effort began shortly.

4.2.6 Trying out more Specifications

As a part of this thesis I, together with Decerno, have tried out the XML v1.0 Specification and the DOM Specification in an application (Pastill). As more specifications reach the W3C Recommendation status it will be necessary to try them out as well. XSL and CSS lie closest at hand to try out; for instance in a look-only version of Pastill.

4.3 Summary

This paper has presented two things, an in-dept overview of emerging XML standards and the ramifications they will have for using XML in modern software development.

XML is a joint effort to create a genuinely open standard, driven entirely by user needs. These needs include:

- *Extensibility,* to define new tags as needed.
- *Structure,* to model data to any level of complexity.
- *Validation,* to check data for structural correctness.
- *Media independence,* to publish content in multiple formats.
- *Vendor and platform independence,* to process any conforming document using standard commercial software or even simple text tools.

Remember that the HTML concept is one of a markup language consisting of a relatively small set of standard tags that are associated with some more-or-less standard behaviors. The XML concept is one of an infinitely large set of possible tags that are associated with no standard behaviors at all. Specification of the behavior has to come from somewhere else. In publishing, that's usually a style sheet, but in other domains it can be something as flexible as JavaBeans or as specialized as an industry-standard protocol around which programmers write standardized applications (such as Pastill).

XML gives us a single, human-readable syntax for serializing just about any kind of structured data, including relational data, in a way that lets it be manipulated and displayed using simple, ubiquitous, standardized tools. The larger implications of a standard, easily processed serial data format are hard to imagine, but they are obviously going to have a large impact on electronic commerce. And it seems clear that electronic commerce is eventually going to become synonymous with commerce in general.

XML can do for data what Java has done for programs, which is to make the data both platform-independent and vendor-independent. This capability is driving a wave of middleware XML applications that will hopefully begin to wash over us during 1999.

As we have proven with the Pastill project, XML can already today be used in an industrial application. The only thing that currently is holding back a more widespread public usage of XML is the lack of tools that support XML. After working with this thesis I am convinced that XML together with Java is the future for distributed software development.

The key to understanding the revolutionary potential of XML is that it is just one piece of a larger picture. XML by itself can provide standardized interchange formats for databases, electronic commerce applications, spreadsheets and much more. This is significant. But XML and XSL together can replace existing word processing and desktop publishing formats as well. It can give us, in effect, a single, completely internationalized format of almost unlimited power for both print and online publishing that is fully interoperable across all products and all platforms. The implications of this go far beyond data exchange and far beyond the Web.

Part 5: Assorted Information

5.1 Abbreviations

ADO	ActiveX Data Objects
ANSI	American National Standards Institute
ANSI X12	American National Standards Institute's Accredited Standards Committee X12 sub-group
API	Applications Programming Interface
ASP	Active Server Pages
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
CSS	Cascading Style Sheets
CSS1	Cascading Style Sheets Level 1
CSS2	Cascading Style Sheets Level 2
DHTML	Dynamic HTML
DOM	Document Object Model
DSSSL	Document Style Semantics and Specification Language
DTD	Document Type Description
EBNF	Extended Bachus-Naur Notation
ECMAScript	Standardization of JavaScript by the European Computer Manufacturer's Association
EDI	Electronic Data Interchange
EEMA	European Electronic Messaging Associations
EJB	Enterprise Java Beans
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HyTime	Hypermedia/Time-based Structuring Language - ISO/IEC 10744
IETF	Internet Engineering Task Force
ISO	International Standards Organization
JVM	Java Virtual Machine
MATH-ML	Mathematical Markup Language
MTS	Microsoft Transaction Server
ODBC	Open Database Connectivity
OQL	Object-orientated Query Language
PDF	Portable Document Format
RFC	Request For Comments
RTF	Rich Text Format
SGML	Standard Generalized Markup Language
SQL	Structured Query Language
STEP	Standard for the Exchange of Product Model Data
UN/EDIFACT	United Nations Standards Messages Directory for Electronic Data Interchange for Administration, Commerce, and Transport
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WFC	Windows Foundation Classes
WIDL	Web Interface Definition Language
WSP	Web Standards Project

WWW	World Wide Web
XLink	XML Link Language
XLL	Extensible Linking Language
XML	Extensible Markup Language
XML-QL	XML Query Language
XPointer	XML Pointer Language
XSL	Extensible Stylesheet Language

5.2 Bibliography

[1] **Extensible Markup Language (XML) Version 1.0**
World Wide Web Consortium Recommendation 10-February-1998
http://www.w3.org/TR/1998/REC-xml-19980210.html

[2] **Cascading Style Sheets Level 1 (CSS1)**
World Wide Web Consortium Recommendation 17-December-1996
http://www.w3.org/TR/REC-CSS1-961217

[3] **Cascading Style Sheets Level 2 (CSS2)**
World Wide Web Consortium Recommendation 12-May-1998
http://www.w3.org/TR/1998/REC-CSS2-19980512/

[4] **Extensible Stylesheet Language (XSL) Version 1.0**
World Wide Web Consortium Working Draft 18-August-1998
http://www.w3.org/TR/1998/WD-xsl-19980818

[5] **Namespaces in XML (XML Namespace**)
World Wide Web Consortium Recommendation 14-January-1999
http://www.w3.org/TR/1999/REC-xml-names-19990114/

[6] **Extensible Linking Language (XLink)**
World Wide Web Consortium Working Draft 3-March-1998:
http://www.w3.org/TR/1998/WD-xlink-19980303

[7] **XML Pointer Language (XPointer)**
World Wide Web Consortium Working Draft 3-March-1998
http://www.w3.org/TR/1998/WD-xptr-19980303

[8] **Level 1 Document Object Model Specification Version 1.0**
World Wide Web Consortium Working Draft 20-July-1998
http://www.w3.org/TR/1998/WD-DOM-19980720/

[9] **Schema for Object-oriented XML (SOX)**
World Wide Web Consortium Note 15-September-1998
http://www.w3.org/TR/1998/NOTE-SOX-19980930/

[10] **XML-QL: A query language for XML**
World Wide Web Consortium Note 19-August-1998
http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/

[11] **Web Interface Definition Language (WIDL)**
World Wide Web Consortium Note 22-September-1997
http://www.w3.org/TR/NOTE-widl-970922

[12] **Vector Markup Language (VML)**
World Wide Web Consortium Note 13-May-1998
http://www.w3.org/TR/1998/NOTE-VML-19980513

[13] **HyperText Markup Language Version 4.0 (HTML)**

World Wide Web Consortium Recommendation 24-Apr-1998
http://www.w3.org/TR/1998/REC-html40-19980424/

[14] **Document Content Description for XML (DCD)**
World Wide Web Consortium Note 31-July-1998
http://www.w3.org/TR/1998/NOTE-dcd-19980731.html

[15] **The XML Handbook**
Charles F. Goldfarb and *Paul Prescod*
ISBN 0-13-081152-1

[16] **XML Complete**
Steven Holzner
ISBN 0-07-913702-4

[17] **Document Style Semantics and Specification Language (DSSSL)**
ISO/IEC standard 10179:1996
ftp://ftp.ornl.gov/pub/sgml/wg8/dsssl/dsssl96b.ps.Z

[18] **XSL Tutorial**
Microsoft
http://www.microsoft.com/workshop/c-frame.htm#/workshop/xml/xsl/tutorial/functions.asp

[19] **XML Workshop**
Microsoft
http://www.microsoft.com/workshop/c-frame.htm#/xml/default.asp

[20] **Component Object Model (COM) technologies Web site**
Microsoft
http://www.microsoft.com/com/default.asp

[21] **Guidelines for using XML for Electronic Data Interchange Version 0.05**
XML/EDI Group 25-January-1998
http://www.geocities.com/WallStreet/Floor/5815/guide.htm

[22] **A Way of Integrating STEP & SGML (V0.4)**
Reiner Reschke, Hugh Tucker 31-August-1996
http://www.eccnet.com/step/White_Paper/

[23] **Product data representation and exchange**
ISO 10303 Industrial automation systems and integration

[24] **Product Data Representation and Exchange**
Part 11: Description methods: The EXPRESS language reference manual
ISO 10303-11 Industrial automation systems and integration

[25] **Product data representation and exchange**
Part 21: Implementation methods: Clear text encoding of the exchange structure
ISO 10303-21 Industrial automation systems and integration

[26] **Data Modeling Report prepared for: W3C XML Specification DTD ("XMLspec")**
World Wide Web Consortium 10-September-1998
http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm

[27] **Querying semi-structured data**
S. Abiteboul
Proceedings of the International Conference on Database Theory, Delphi, Greece, January 1997.
http://www-rocq.inria.fr/~abitebou/pub/icdt97.semistructured.talk.ps

[28] **Query Optimization for Semi-structured Data**
J. McHugh and *J. Widom*

Technical report, November 1997
http://db.stanford.edu/pub/papers/qo.ps

[29] **Mathematical Markup Language (MathML) 1.0 Specification**
World Wide Web Consortium Recommendation 7-April-1998
http://www.w3.org/TR/1998/REC-MathML-19980407/

[30] **Rules for Electronic Data Interchange for Administration, Commerce, and Transport (EDIFACT)**
United Nations
http://www.unece.org/trade/untdid/welcom1.htm

[31] **ActiveX Data Objects (ADO)**
Microsoft
http://www.microsoft.com/data/ado/sams/ch08.htm

[32] **Data International Standards Association (DISA)**
Accredited Standards Committee X12 (ASC12)
http://polaris.disa.org/

[33] **Amaya - W3C's Browser/Editor**
World Wide Web Consortium
http://www.w3.org/Amaya/

[34] **XML Support in Oracle8*i* and Beyond**
An Oracle Technical Whitepaper, 9-November-1998
http://www.oracle.com/xml/documents/xml_twp/

[35] **XML: The future of EDI?**
Uche Ogbuji
Sunworld – February 1999. Vol. 13 No.2
http://www.sunworld.com/swol-02-1999/swol-02-xmledi.html

[36] **SAX 1.0: The Simple API for XML**
http://www.megginson.com/SAX/

[37] **DCOM and CORBA Side by Side, Step by Step, and Layer by Layer**
P.Emerald Chung, Yennun Huang and *Shalini Yajnik*
Bell Laboratories, Lucent Technologies
http://www.cs.wustl.edu/~schmidt/submit/Paper.html

[38] **Dynamic HTML in Action**
Willian J. Pardi and *Eric M. Schurman*
ISBN 1-57231-820-1

[39] **CORBA/IIOP 2.2 Specification**:
Object Management Group (OMG) 1-July-1998
http://www.omg.org/corba/corbaiiop.html

[40] **The Component Object Model: Technical Overview**
Microsoft
http://www.microsoft.com/com/wpaper/Com_modl.asp

[41] **Distributed Component Object Model Protocol DCOM/1.0**
Microsoft
http://premium.microsoft.com/msdn/library/specs/dcom/
distributedcomponentobjectmodelprotocoldcom10.htm

[42] **Microsoft Transaction Server White Papers**
Microsoft
http://www.microsoft.com/ntserver/appservice/techdetails/techspecs/mtswp.asp

[43] **The SGML Handbook**
Charles F. Goldfarb
ISBN: 0-19-853737-1

[44] **Hypermedia/Time-based Structuring Language (HyTime)**
ISO/IEC 10744-1992(E)
http://www.ornl.gov/sgml/wg8/docs/n1920/html/n1920.html

[45] **Guidelines for Electronic Text Encoding and Interchange**
C. M. Sperberg-McQueen and *Lou Burnard*
Association for Computers and the Humanities (ACH), Association for Computational Linguistics (ACL), and Association for Literary and Linguistic Computing (ALLC).

[46] **Uniform Resource Locators**
Internet Engineering Task Force (IETF). RFC 1738 1991.
http://www.w3.org/Addressing/rfc1738.txt

[47] **Relative Uniform Resource Locators**
Internet Engineering Task Force (IETF). RFC 1808 1995.
http://www.w3.org/Addressing/rfc1808.txt

[48] **ENTERPRISE JAVABEANS™ TECHNOLOGY Server Component Model for the Java Platform**
JavaSoft
http://www.javasoft.com/products/ejb/white_paper.html

[49] **Accessing the World of Information: Open Database Connectivity (ODBC)**
Microsoft
http://premium.microsoft.com/msdn/library/backgrnd/html/msdn_odbcbg.htm

[50] **An ASP You Can Grasp: The ABCs of Active Server Pages**
Microsoft
http://www.microsoft.com/workshop/c-frame.htm#/workshop/server/default.asp

[51] ***Uniform Resource Identifiers (URI): Generic Syntax and Semantics***
Berners-Lee, T., R. Fielding, and *L. Masinter*. 1997
(Work in progress; see updates to RFC1738.)

[52] **XML-Data**
World Wide Web Consortium Note 05-Jan-1998
http://www.w3.org/TR/1998/NOTE-XML-data-0105/

[53] **ECMAScript Language Specification**
Standard ECMA-262 2nd edition. June 1998
http://www.ecma.ch/stand/ecma-262.htm

[54] **Chemical Markup Language**
http://ftp.sunet.se/ftp/pub/www/utilities/mapmarker/cml/

[55] **The JUMBO browser**
The Virtual School of Molecular Science
http://ala.vsms.nottingham.ac.uk/vsms/java/jumbo/

[56] **Windows Foundation Classes (WFC)**
Microsoft
http://premium.microsoft.com/msdn/library/periodic/period98/html/vji0898c.htm

[57] **XML Parser for Java**
IBM AlphaWorks

http://www.alphaworks.ibm.com/formula/XML

[58] **Java XML Parsers – A comparative Evaluation of 7 Free Tools**
Juancarlo Añez
Java Report Online
http://www.javareport.com/html/products/prod_rev.shtml

[59] **A Technique for High Performance Data Compression**
Terry A. Welch
IEEE Computer, Vol. 17, No. 6, 1984, pp. 8-19

[60] **The Lorel Query Language for Semistructured Data**
S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener
International Journal on Digital Libraries, vol. 1, no. 1, 4/1997

[61] **A query language and optimization techniques for unstructured data**
Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu
Proceedings of ACM-SIGMOD International Conference on Management of Data, 1996.

[62] **A Query Language for a Web-Site Management System**
Mary Fernandez, Daniela Florescu, Alon Levy, and Dan Suciu
SIGMOD Record, vol. 26, no. 3, 9/1997

[63] **Literate Programming**
Henrik Turbell
Image Processing Group, Department of Electrical Engineering, Linköping University
March 1997
http://www.isy.liu.se/~turbell/litprog/