GigaTS

XML

Hype or hope?

Telematica Instituut

Colophon

Date :	November 20th, 2000
Version :	2.0
Change :	Final version
Project reference :	GigaTS/D2.2.4
TI reference :	TI/RS/2000/038
URL :	https://extranet.telin.nl/docuserver/dscgi/ds.py/Get/File-11272
Access permissions :	Public
Status :	Final
Editor :	Johan Koolwaaij
Company :	Telematica Instituut
Authors :	Wil Janssen
	Johan Koolwaaij
	Milena Stefanova

Synopsis:
Few internet technologies attract as much attention as XML does. It is said to be the lingua franca *for Internet. In this report we describe XML, give examples of its use, and sketch the developments in XML.*

Acknowledgements:
This work was done within the Giga Transaction Services Project (http://gigats.gigaport.nl/). GigaTS supports organizations in the development of innovative transaction services. It does so with state of the art knowledge, methods and software tools that allow for effective development of new services. The editor would like to thank Frank Biemans, Hugo ter Doest, Diederik van Leeuwen and Petra van der Stappen for their comments and useful suggestions.

Table of Contents

1 Introduction

1.1 XML: hype or hope?

Few internet technologies have attracted the amount of attention that XML has. It is called the *lingua franca* of the Internet and the follow-up of EDI. No self-respecting e-business software provider can afford not to work with XML. Many companies are looking at the possibilities of XML for them. This document is intended for those enterprises in the e-business community that want to gain more in-depth knowledge about XML and its application in business-to-business document exchange.

In this report we want to provide some insight in what XML is and what it can do for you. We go into details on what XML is, and on how to handle XML. In the second part of this document we describe a number of applications of XML in finance, trade and multimedia. Also, we discuss generic frameworks to support XML and XML document exchange. This overview is by no means complete. The pace at which new standards and applications are being developed is too high to allow for completeness. Yet, the applications presented represent most of the important developments worldwide.

In doing so we try to resolve an important question: is XML primarily hype, and will it soon be forgotten, or does it really bring something to the e-business community?

1.2 The essence of XML

Extensible Markup Language (XML) is a simple and flexible markup language derived from the Standard Generalized Markup Language (SGML, ISO 8879). SGML is the mother of all markup languages, and XML can be seen as SGML without those parts that are hard to implement in software products. Designed to meet the challenges of large-scale electronic publishing, XML also plays an increasingly important role in the exchange of a wide variety of data on the Web. The main characteristics of XML are:

- enabling internationalized media-independent electronic publishing;
- allowing industries to define platform-independent protocols for the exchange of data, especially the data of electronic commerce;
- delivering information to user agents in a form that allows automatic processing after receipt;
- making it easy for people to process data using inexpensive software;
- allowing people to display information the way they want it;
- providing metadata -data about information- that will help people find information and help information producers and consumers find each other.

Basically, XML is very simple. It is a so-called markup language, which separates the structure, content, and layout of documents, as opposed to HTML, which is primarily a formatting language. HTML is loosely based on SGML and can be seen as SGML with exactly one fixed document type definition (DTD), which is focused on the formatting of documents. In XML, the user can specify the document structure in a user-defined DTD, which is extensible. Structure and content are strictly separated, hence a DTD can be used for several XML document as a template. Finally, the presentation is specified in a stylesheet. Using different stylesheets, the same content can be presented in numerous ways, without having to reorganize the same content.

These are the primary ingredients: a document type definition and a presentation specification that states how the information in an XML document should be displayed. Its simplicity, however, also is its power: instead of having a fixed language to specify the content of files on the World Wide Web such as HTML, XML gives the possibility to define new languages, new formats. These can be used for many different purposes, ranging from specifying configuration files in servers to EDI messages to Common Business Objects. Many initiatives are under way to standardize different language formats in XML, of which the most prominent ones are discussed below.

1.3 Overview and other resources

As XML development goes at an extreme pace, it is a good idea to regularly check upon the most important XML resources:

- The W3C XML page (http://www.w3.org/XML.htm)
- The IBM XML resources page (http://www.ibm.com/developer/xml) provides a nice XML tutorial as well (http://www.ibm.com/software/developer/education/xmlintro).
- XML.com (http://xml.com) is kind of a portal for XML development and includes a good technical introduction to XML (http://xml.com/pub/98/10/guide0.html). Also, a concise overview of all XML standards is presented there (http://xml.com/pub/stdlist).
- The Internet Engineering Task Force, IETF (http://www.ietf.org) has a number of XML standards under its supervision as well, such as the Internet Open Trading Protocol.

In this report we give an account of the XML features and important industry standards based on XML. In Section 2 some history behind XML is presented. Section 3 describes the basic structure of XML documents and DTDs, while Section 4 contains information about advanced XML features such as Schemas, Style Sheets and Namespaces. Section 5 contains a classification of the basic tools used to process XML documents, such as parsers and validators. Readers who are less interested in the details about the structure of XML documents may skip the sections 3.2 - 3.4 and proceed with chapter 4.

Sections 6 through 8 give short summaries of industry based standards in b2b document exchange. We elaborate on XML standards in financial and trading sectors, as well as multimedia information disclosure. Section 9 describes general frameworks, XML middleware, which can be used for solving the problems of interoperability.

2 History of XML

XML comes from a rich history of text processing systems. The first wave of automated text processing was computer typesetting. Authors would type in a document and describe how they would like it o be formatted. The computer would print out a document with the described text and formatting. The file format containing the mix of the actual data in a document plus the formatting rules is called a rendition. Examples of rendition notations are RTF and LaTeX. Typesetting notations are often based on formatting markup. Formatting markup languages contain sets of tags, which can be interpreted by a computer while formatting the document. LaTeX is an example of a formatting markup language.

Around the late sixties there was a need to do more things with documents than just formatting them. For example, documents can be stored, managed or published. For this purpose the *Standard Generalized Markup Language* (SGML) was created. A generalized markup language can be understood by a variety of applications, since it distinguishes between structure (abstraction), content and representation of documents. In 1969 the IBM team developed a language called the *Generalized Markup Language.* Later, the concept of “validating parser” was created and that was the beginning of standardizing this language. The SGML was approved as an ISO standard in 1986. At that time SGML had become a large and complex language, which was the de facto standard for interchange of complex documents.

In 1989, a researcher named Tim Berners-Lee proposed that information could be shared within the CERN European Nuclear Research Facility using hyperlinked text documents. He was advised to use a SGML-based syntax. A simple hypertext version of SGML was developed in this way. It was called the *Hypertext Markup Language* (HTML), see [HTML]. HTML inherited some important strength from SGML. Many of its element types were generalized and descriptive formatting constructs, unlike the fixed formatting of Tex and LaTeX. This meant that HTML documents could be displayed on text screens, under graphical user interfaces, and even spoken using audio interfaces. In addition, HTML was widely adopted due to its simplicity. This led to standardizing the HTML language by the World Wide Web Consortium (W3C) in 1995. On the other hand HTML uses a fixed set of element types. It is not extendable and moreover it was not rigorously defined until years after its invention. This made it possible for competing vendors to extend it in arbitrary ways for their software. In this way incompatible versions of HTML were created. Most of the extensions were formatting commands and thus damaged the Web’s interoperability. Some formatting extensions were invented to actually represent abstraction extensions. For example, particular HTML developers would decide to use superscripts to indicate footnotes. The abstraction information behind that formatting rule is lost if the document is to be translated into another representation. So new documents representing the same information threatened to flood the Web.

As the interoperability and scalability of the Web became more and more endangered by proprietary mark-up, the W3C decided to act. They turn back to some ideas from the SGML language. First, they invented a simple HTML-specific stylesheet language, called *Cascading Style Sheets* (CSS), see [CSS], that allowed people to attach formatting to HTML documents without filling the HTML itself with proprietary mark-up. Next, a simple mechanism for adding abstractions to HTML was developed and called Extensible HTML (XHTML). This allowed abstractions to be added to the HTML language, but it did not constrain their occurrence.

The two extensions above brought HTML back to being a standard. Still it did not incorporate an important cornerstone of SGML, namely, that document types should be formally defined so that the documents can be checked for validity against them. Therefore, the W3C decided to develop a subset of SGML that would preserve the main ideas behind SGML, but also stay simple and usable for the Web: the *Extensible Markup Language* (XML).

The first phase of the XML Activity, started in June 1996, culminated in the W3C XML 1.0 Recommendation, issued February 1998. In the second phase, work proceeded in a number of working groups in parallel, resulting in the January 1999 Recommendation Namespaces in XML, the June 1999 Recommendation for Style Sheet linking, and the November 1999 Recommendation for Extensible Style Sheet (XSL) Transformations. A substantial number of other XML working groups have produced various drafts and notes. The most important are those related to XML Schema Language and XSL. The XML Schema language is conceived as an extension of the XML mechanism for specifying document types. XML Schema has recently (in October 2000) been advanced to Candidate Recommendation status.

In September 1999, the third phase of the XML activity began, continuing the unfinished work from the second phase and introducing a Working Group on XML Query and plans for a Working Group on XML Packaging, see [XML].

Parallel with the W3C activities various industries started XML-related activities aiming at standardizing processes, concepts and formats used in such industries. Several of these industry standards are described in the sequel. The creation of separate industry standards leaded to conceiving of platforms, which allow interoperation between different industries. Such platforms aim at solving interoperability problems in software in a general scale. Examples are Eco, Biztalk, and ebXML (also described in the sequel). The ideas behind such platforms and behind the original XML standard are extended in the new initiative of the W3C for a semantic web [SemWeb]. The goal of the semantic web initiative is to increase the interoperability, scalability and to some extent the security of software downloadable via the Web.

3 Overview of XML

XML is a markup language. Historically, the editor placed markup in a text to describe how the text should be laid out. Nowadays, markup defines the meaning of a text. Usually, tags are used, which enclose a text region, to define the meaning of that text region. In HTML, for example, the bold tag was used to denote bold text, e.g. <b>this text is bold</b> looks like **this text is bold** in a HTML browser. But where HTML was focused on formatting of a document, XML is focused on the meaning and structure of a document. For example, tags can be used to denote someone's name, with a specified first and last name: <name><firstname>Peter</firstname><lastname>Jacob</lastname></name>. In this example, it is clear that Jacob is the last name, and not the first name. This way, XML documents are automatically interpretable.

3.1 Basic XML

The basic idea of XML is that it separates actual data from its structure and from the way it is processed (in particular presented). XML was created with the thought to be used for document processing systems, and that is why the primary data in XML are various documents. Recently, XML has been extended to include arbitrary data types. The structure of an XML document is defined as a document type. A document type is an abstraction representing a class of documents with common structure and meaning. Document types can be seen as simplifications of what is called "abstract data types" in various programming languages. Since document types are abstractions of content (e.g. describe some intended meaning or structure), they are not supposed to carry information about how the content is to be rendered (unless the intended meaning of the content is a format description). The actual document content is represented as an XML document instance, which is structured with the help of user-defined tags. These tags are defined in a document type definition (DTD) or directly in the document instance. In the first case the document is declared to be of a certain type. Such structured documents are easily processed in order to be presented to users or to be supplied to software applications for further processing.

This is illustrated by the following example. Assume we have an insurance company that wants to exchange documents using mail containing the results of a car damage claim. The mail message will contain information about the sender (i.e. the insurance company itself), the receiver (i.e., the insurant), and the actual decision made regarding a specific claim. An XML document containing this information is presented below.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="claimdecision.xsl"?>
<!DOCTYPE CLAIMDECISION SYSTEM "claimdecision.dtd">

<CLAIMDECISION>
        <SENDER>
                <NAME>Middelman Verzekeringen</NAME>
                <ADDRESS>
                        <STREET>Raadhuisplein 14</STREET>
                        <ZIP>8911 PP</ZIP>
                        <TOWN>Leewarden</TOWN>
                </ADDRESS>
        </SENDER>
        <RECEIVER>
                <POLICYNR>123456111222</POLICYNR>
                <NAME>Anne Wijnands</NAME>
                <ADDRESS>
```

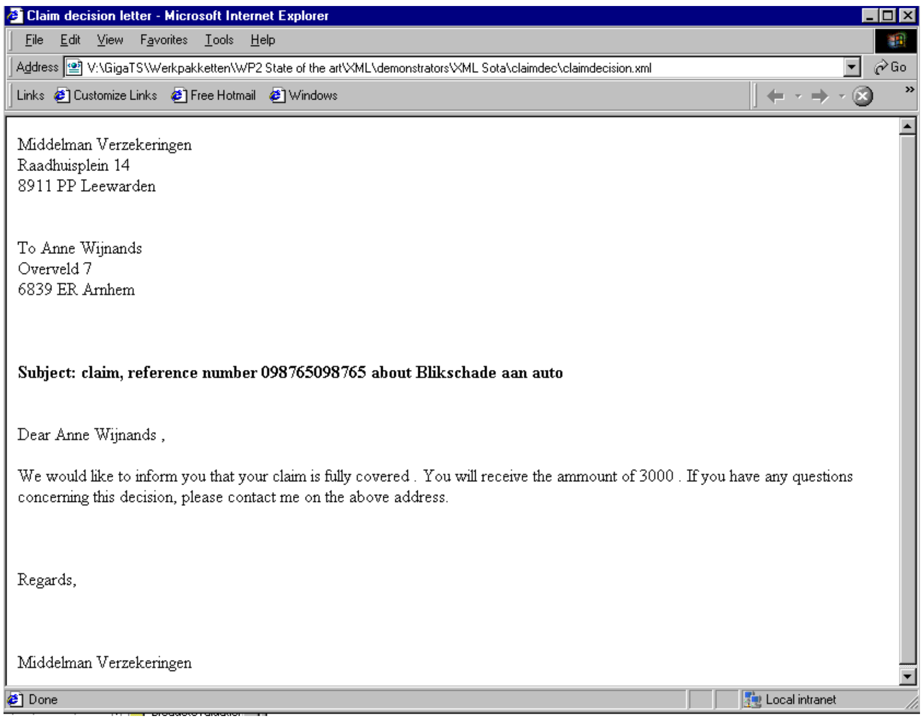
```
                        <STREET>Overveld 7</STREET>
                        <ZIP>6839 ER</ZIP>
                        <TOWN>Arnhem</TOWN>
                </ADDRESS>
        </RECEIVER>
        <CLAIM>
                <CLAIMREFNR>098765098765</CLAIMREFNR>
                <CLAIMDESCRIPTION>Blikschade aan auto</CLAIMDESCRIPTION>
        </CLAIM>
        <DECISION>
                <STATUS>grant</STATUS>
                <AMOUNT>3000</AMOUNT>
                <INFO>fully covered</INFO>
        </DECISION>
</CLAIMDECISION>
```

This XML document contains a document type declaration saying that the document instance should match the document type defined in the file c*laimdecision.dtd*. This is a text file, which is defined in the following way.

```
<!ELEMENT CLAIMDECISION   (SENDER, RECEIVER, CLAIM, DECISION)>
<!ELEMENT SENDER           (NAME, ADDRESS)>
<!ELEMENT RECEIVER         (POLICYNR, NAME, ADDRESS)>
<!ELEMENT CLAIM            (CLAIMREFNR, CLAIMDESCRIPTION)>
<!ELEMENT DECISION         (STATUS, AMOUNT, INFO)>
<!ELEMENT NAME             (#PCDATA)>
<!ELEMENT ADDRESS          (STREET, ZIP, TOWN)>
<!ELEMENT STREET           (#PCDATA)>
<!ELEMENT ZIP              (#PCDATA)>
<!ELEMENT TOWN             (#PCDATA)>
<!ELEMENT POLICYNR         (#PCDATA)>
<!ELEMENT CLAIMREFNR       (#PCDATA)>
<!ELEMENT CLAIMDESCRIPTION (#PCDATA)>
<!ELEMENT AMOUNT           (#PCDATA)>
<!ELEMENT STATUS           (#PCDATA)>
<!ELEMENT INFO             (#PCDATA)>
```

Note that XML documents can be created without being declared of a specific DTD (and thus without the need to write DTDs). However, DTDs restrict the arbitrary usage of tags and are very useful in cases where two or more parties have agreed on certain structure for document exchange. If a DTD is used, it can be checked automatically whether certain documents conform to the structure prescribed in the DTD. As an example, the HTML language has been specified as a DTD. If an HTML document is declared to be of this DTD, then it should use the standard HTML syntax, otherwise it will be ruled out as being an invalid HTML document [HTMDTD].

Going back to the example about the claim decision letter, one can notice that there is a reference to a style sheet file (*claimdecision.xsl*) which has to be applied to the document instance in order to obtain a suitable rendition of the document. The document rendition displayed in a browser may look like the following picture.


Claim decision letter - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address V:\GigaTS\Werkpakketten\WP2 State of the art\XML\demonstrators\XML Sota\claimdec\claimdecision.xml
Go
Links Customize Links Free Hotmail Windows
Done
Local intranet

Middelman Verzekeringen
Raadhuisplein 14
8911 PP Leewarden

To Anne Wijnands
Overveld 7
6839 ER Arnhem

Subject: claim, reference number 098765098765 about Blikschade aan auto

Dear Anne Wijnands ,

We would like to inform you that your claim is fully covered . You will receive the ammount of 3000 . If you have any questions concerning this decision, please contact me on the above address.

Regards,

Middelman Verzekeringen

The XSL file used for displaying the claim decision XML-document is partially shown below.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
        <html>
                <head>
                        <title>Claim decision letter</title>
                </head>
                <body>
                        <xsl:apply-templates select="CLAIMDECISION"/>
                </body>
        </html>
</xsl:template>
…
<xsl:template match="DECISION">
        We would like to inform you that your claim
        is
        <xsl:value-of select="INFO" />
        .
        You will receive the ammount of
        <xsl:value-of select="AMOUNT" />
        . If you have any questions concerning this decision, please contact me on
the above address.
</xsl:template>

<xsl:template match="ADDRESS">
        <br/>
        <xsl:value-of select="STREET" />
        <br/>
        <xsl:value-of select="ZIP" />
        <xsl:value-of select="TOWN" />
</xsl:template>

</xsl:stylesheet>
```

Till the end of this chapter we will describe the basics of XML in more details. These are the XML language itself and the language for defining DTDs. The section finishes with some basics about processing of XML documents. The style sheet language designed for use with XML documents will be described in more detail in section 4, XML Extensions. More about XML processing can be found in section 5. Reader not interested in the details of XML may skip this part, and proceed to chapter 4.

3.2 Structure of an XML document

An XML document contains two parts, a *prolog* and a *document instance*. The prolog contains meta information about the XML document instance, such as version of XML used, type of the document instance or the way it should be further processed. The prolog part is optional. The document instance specifies the content of the XML document. This content is structured with the help of user-defined tags. The tags contain meta-information about the actual document content represented by the XML document instance. The structure imposed on the document is used in the specification of the document rendition semantics or for defining other computational semantics of the document.

3.2.1 Prolog

The prolog part of an XML document may contain an *XML declaration, Document Type Declaration* and/or one or more *Processing Instructions.* The XML declaration specifies the version of XML used. For example,

```
<?xml version="1.0"?>
```

The processing instructions are instructions to applications which may further process the XML document instance. An example is the style-sheet processing instruction presented below.

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

Processing of XML documents will be considered in more detail in section 3.4.

The prolog part of an XML document may contain a document type declaration as well. The document type declaration states that the document instance contained in the XML document is of certain type. This type can be defined as an external file (entity) with file extension *.dtd*, can be defined internally as part of the document type declaration or can contain mixed external and internal content. Document type definitions (DTDs) are considered in more detail in section 3.3. An example of a type declaration where the type itself is defined externally is shown below.

```
<!DOCTYPE CLAIMDECISION SYSTEM "claimdecision.dtd">
```

In the example above the document instance is of type *CLAIMDECISION*. This type is defined in the file "*claimdecision.dtd*". The type *CLAIMDECISION* can be defined internally as shown below.

```
<!DOCTYPE CLAIMDECISION [
        <!ELEMENT CLAIMDECISION (SENDER, RECEIVER, CLAIM, DECISION)>
        <!ELEMENT SENDER (#PCDATA)>
        <!ELEMENT RECEIVER (#PCDATA)>
        <!ELEMENT CLAIM (#PCDATA)>
        <!ELEMENT DECISION(#PCDATA)> ] >
```

3.2.2 Document Instances

XML document instances are composed of markup and content. There are five kinds of markup that can occur in an XML document instance: *elements*, *entity references*, *comments*, *processing instructions*, and *marked sections*. The following sections introduce some of these markup concepts.

Elements are the most common form of markup. Delimited by angle brackets, most elements identify the nature of the content they surround. Some elements may be empty, that is, they have no content. If an element is not empty, it begins with a start-tag, e.g., <*name*>, and ends with an end-tag, e.g., <*/name*>.

Attributes are name-value pairs that occur inside start-tags after the element name. For example, <*div class="preface"*> is a div element with the attribute class having the value preface. In XML, all attribute values must be quoted.

Entities in XML correspond to macros in programming languages. Each entity has a name and content. Entities are referenced by name in XML documents and each occurrence of an entity name is replaced by the corresponding content before processing XML documents.

The syntax of an entity reference is *&name* where *name* denotes the name of the referenced entity. Entity references can occur at any place in an element content specification. Some of the entities are predefined in XML. They are used to represent special characters, such as <.

Non-predefined entities should be defined and declared in a DTD first. Then they can be referenced in XML document instances of the DTD specified. Entity references to such entities are used to represent text which repeats or varies often, or to insert non-textual content into XML documents (for more details see section 3.3.3).

Processing instructions (PIs) are an escape hatch to provide information to an application. Like comments, they are not textually part of the XML document, but the XML processor is required to pass them to an application. Processing instructions have the form: *<?name pidata?>*. The name, called the PI target, identifies the PI to the application. Applications should process only the targets they recognize and ignore all other PIs. We elaborate on processing instructions in section 3.4.

In a document, a CDATA section instructs the parser to ignore most markup characters. Consider a source code listing in an XML document. It might contain characters that the XML parser would ordinarily recognize as markup (< and &, for example). In order to prevent this, a CDATA section can be used.

3.3 Document Type Definitions (DTDs)

The structure of an XML document instance can be defined as a Document Type Definition (DTD). If a document instance is declared to be of a certain document type (see section 3.2.1), then while (pre-) processing this XML document some validity checks can be done to assure that the document instance matches the structure defined as a DTD. Basically a DTD is composed of *element type declarations*, *attribute declarations*, *entity declarations* and *notation declarations.* These declarations are done with the help of XML predefined mark-up tags. In addition DTDs may contain references to some entities and conditional sections.

3.3.1 Element type declarations

Element type declarations are the core of a DTD. Simple DTDs are composed only from such type declarations for elements of a class of XML documents. Element types are declared with the help of the XML tag *ELEMENT* followed by the name of the element type and a specification of the type itself. The whole element type declaration is placed between <! and >. For example,

```
<! ELEMENT NAME (#PCDATA)>
```

is a simple element type declaration. The type declared has a name *NAME* and in fact it is defined as a string of characters (the word *#PCDATA* stands for Parsed Character Data and is a string of characters). Except for strings of characters, XML allows to define empty types, types which can have any content, types which are a non-deterministic choice of a finite number of types, record types (whose elements are sequences of elements of other types), non-empty lists, lists (possibly empty) and optional types. The syntax of all these constructions is presented in Table 1.

Element Type	Syntax
Character string	`(#PCDATA)`
Empty	`EMPTY`
Any content	`ANY`
Non-deterministic choice (an element of this type is either of type T1, or T2, …, or Tn)	(T1 | T2 | … | T2)
Record (an element is an ordered sequence of elements of types T1, T2, …, Tn)	(T1, T2, …, Tn)
Non-empty list (an element is a non-empty finite sequence of elements of type T)	T+
List (an element is any finite sequence of elements of type T	T*
Optional type (an element is a sequence of 0 or 1 elements of type T)	T?

Table 1. DTD Element Type Specifications

3.3.2 Attribute declarations

Element types can be further refined by specifying their attributes. For example, consider that we want to specify a letter document. A letter document would be characterized by a sender, a receiver and a letter body.

```
<!ELEMENT LETTER (SENDER, RECEIVER, BODY)>
<!ELEMENT SENDER (PARTY_INFO)>
<!ELEMENT RECEIVER (PARTY_INFO)>
<!ELEMENT PARTY_INFO (NAME, ADDRES)>
```

Imagine now that we want to speak only about letters exchanged between an insurance company and an insurant. It may be convenient to add an attribute called *party_type* to the element type *PARTY_INFO*, instead of defining two new element types for the insurance company and the insurant which would contain the same structured (i.e., name and address) data. This attribute can be defined as follows.

```
<! ATTLIST PARTY_INFO party_type CHOICE (INSURANCE_COMPANY | INSURANT) #REQUIRED >
```

An example of the party_info element could be

```
<PARTY_INFO party_type = ”INSURANT”>
  <NAME> BOB JONSON </NAME>
  <ADDRESS> VIAWEG 15 ENSCHEDE </ADDRESS>
</PARTY_INFO>
```

Attributes of element types are defined with the help of the XML pre-defined tag *ATTLIST* followed by the element type name and a list of attribute specifications for this element type. An attribute specification consists of the name of the attribute, its type and a default value or indication whether the attribute is required or not. Table 2 summarizes the attribute types.

Attribute Type	Syntax	Meaning
Character Data	`CDATA`	String of arbitrary characters
Name Token	`NMTOKEN`	String of characters which are letters, numbers and a select group of specialized characters (XML name tokens).
Notation	`NOTATION`	Valid XML names, which moreover has been declared as notations (see section 3.3.4).
Entity	`ENTITY`	Names of unparsed entities (see section 3.3.3).
Identifier	`ID`	An XML name which may appear at most once in a document.
Reference	`IDREF`	An XML name which must appear as a value of an attribute of type ID.
List	`NMTOKENS` `ENTITIES` `IDREFS`	There are lists of values of type `NMTOKEN`, `ENTITY` and `IDREF` only.
Enumeration	`CHOICE` (op1 | op2 | … |opn)	This is a finite set of values, op1, op2, …, opn, where each of them must be a name token

Table 2. Attribute types

There are three kinds of attributes, namely required, optional, and fixed.

3.3.3 Entity declarations and entity references

As it was mentioned in section 3.2.2 non-predefined entities should be declared first. Declaring an entity means assigning a name to its content. The entity content itself can be defined inside the declaration or in a separate file.

There are several syntactical notations for declaring entities, which reflect the usage of the different entities, different content and location of content. According to their usage entities are classified into *general* and *parameter* entities. General entities are referenced inside XML document instances. They abbreviate parts of document instances or non-XML content to be inserted into the document during processing. Parameter entities on the other hand are referenced inside DTDs. They abbreviate parts of DTDs.

General entities

The general entities are declared in the following way.

<!ENTITY *name content_specification*>

The general entities are further classified according to their content into *parsed* and *unparsed*. Parsed entities contain text which is well-defined XML element content. Unparsed entities contain arbitrary data, such as image, text, voice, etc. Parsed entities can be *internal* or *external* depending on whether the entity content is defined inside the entity specification or in an external file. The syntax of a general parsed internal entity is the following.

<!ENTITY *name* "*XML element content* ">

The external entities are located through external identifiers, which can be system or public. System identifiers are composed by the reserved word SYSTEM followed by a URI (Uniform Resource Identifier). Public identifiers are composed by the reserved word PUBLIC followed by a world-wide unique name of the file where the entity is stored. Some examples of internal and external parsed entities are presented below.

```
<!ENTITY xml "Extensible Markup Language">
<!ENTITY accident "<ERROR> Error </ERROR>" >
<!ENTITY entity SYSTEM "http://www.telin.nl/xml/entity.xml">
```

General parsed entities can be referred anywhere inside an XML element content specification or inside an attribute value.

General unparsed entities hold data such as images or text in some data object notation. They are defined using the following syntax.

<!ENTITY *name external_identifier* NDATA *notation_name*>

The external identifier is defined as above. It is followed by the reserved word NDATA and by the name of the data object notation. This name should be a name of declared notation (see section 3.3.4). An example for a general unparsed entity is the following.

```
<!ENTITY picture SYSTEM "http://www.telin.nl/my_picture.gif" NDTA GIF>
```

Unparsed entities can be referred in the values of attributes of type ENTITY or ENTITIES.

Parameter entities

As it has been mentioned before parameter entities abbreviate parts of DTDs. The content of such entities is a list of well-defined element type declarations or attribute declarations. Parameter entities are defined in the following way.

<!ENTITY %*name content_specification*>

The content specification can contain the full meaning of the entity name, or it can be an external identifier of declarations stored in a file. Parameter entities are referenced inside the DTDs using %*name*. As an example, assume we want to define two separate classes of XML documents representing a general letter and a letter from an insurance company to its insurants informing them about decisions regarding claims. We may define an entity in a file `letter.ent`, whose content is described below.

```
<!ELEMENT LETTER (SENDER, RECEIVER, BODY)>
<!ELEMENT SENDER (PARTY_INFO)>
<!ELEMENT RECEIVER (PARTY_INFO)>
<!ELEMENT PARTY_INFO (NAME, ADDRESS)>
```

A DTD representing the class of general letter document can be defined in the following way.

```
<!ENTITY letter SYSTEM "letter.ent">
%letter
<!ELEMENT BODY (#PCDATA)>
<!ELEMENT NAME (FIRST, FAMILY)>
<!ELEMENT ADDRESS (#PCDATA)>
...
```

3.3.4 Notation declarations

Notations are used for describing data content. A *data content notation* (or notation for short) is a definition of how data of specific type should be interpreted. For example, XML is a notation. A notation declaration in XML assigns an internal name to an existing notation so that it can be referred to in attribute declarations, unparsed entity declarations, and processing instructions (see section 3.4). The general form of a notation declaration is

<!NOTATION *name specification*>

The notation specification is an external identifier for documentation, formal specification or applications, which can handle objects represented in the notation. Examples of notation declarations are the following.

```
        <!NOTATION HTML SYSTEM "http://www.w3.org/Markup">
        <!NOTATION GIF SYSTEM "gifmagic.exe">
```

3.4 Processing of XML documents

As it has been mentioned in section 3.2.1, XML documents can contain instructions to applications, which can process these documents. Such instructions are called *processing instructions.* There is no precise syntax for processing instructions, it basically depends on the application toward which a processing instruction is targeted. The processing instructions are put between "`<?`" and "`?>`" and contain the name of the target application right after the "`<?`". Often processing instructions contain pairs of attribute names and values associated with the named application. Examples of processing instructions are instructions to a style sheet application (see section 4.3 for more details).

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
<?xml-stylesheet href="mystyle.xsl" type="text/xsl"?>
```

The first processing instruction in the examples above is for a Cascading Style Sheet (CSS) processor application and the second is for an Extensible Style Sheet (XSL) processor application.

It is possible to use other than formatting applications to process XML documents. For example one can use a software component to further process an XML document.

```
<?server-component method=”submitOrder” ?>
```

In the example above, the XML document will be further processed by an application called *server-component* by invoking its method *submit order*.

4 XML Extensions

4.1 Name spaces

In XML 1.0, the names of element types, attributes and targets of programming instructions are local for documents and DTDs. This is a problem in distributed environments, where documents and DTDs can be composed from documents and DTDs created by other people. The W3C has come with a solution to this problem, called namespaces. Namespaces are an approved recommendation of W3C from January 1999. Namespaces provide a way for qualification of names of elements and attributes. This is done by adding a prefix to the name of the element. The prefix in a name is associated with a namespace. A namespace is a unique identifier associated with people, companies etc. Due to their uniqueness XML uses URLs to express namespaces. Since URLs are not valid XML names, a prefix is associated with them and further used in the definition of documents. Name spaces are declared in the `xmlns` attribute, which is an optional attribute for elements. Namespace declarations are scoped by their declaring elements, i.e., they apply to the element, its children, the children of its children etc. If namespaces are declared in the root element they apply to the whole document. As an example consider the following.

```
<claim:CLAIMDECISION xmlns:claim="http://www.telin.nl/claims" >
 <claim:SENDER>
    <NAME>Middelman Verzekeringen</NAME>
    <ADDRESS>
      <STREET>Raadhuisplein 14</STREET>
      <ZIP>8911 PP</ZIP>
      <TOWN>Leewarden</TOWN>
    </ADDRESS>
 </claim:SENDER>
…
</claim:CLAIMDECISION>
```

The elements whose names have no prefix "`claim`" in the example above are considered local for the document.

Note that namespaces and DTDs do not work together. This is because DTDs have been designed with the purpose to serve non-distributed documents and documents conforming to one DTD with no parts which would conform to some other separate types. Because of this and because the DTD language has limited expressive power, the W3C has come up with a specification of an XML Schema language. This language will be considered in the following section.

4.2 XML Schema

The XML Schema language extends and generalizes the DTD language. A schema is a model of describing the structure and sometimes the semantics of information. In fact, the DTD language is a schema language borrowed from SGML. The XML Schema language is the first attempt to replace DTD with something better. Currently the XML Schema 1.0 language is a working draft of the W3C. It has been expected to become a candidate recommendation from June 2000, but is still in a working draft version.

So, why are DTDs not a suitable schema language? The first emerging problem is the syntax. The syntax for XML documents and DTDs are different, so existing XML tools cannot be used to check the validity of the DTD syntax. Another, more essential problems are the semantics. DTDs allow us to express very limited semantical information about the documents. That is because DTDs make use of few data types and provide little flexibility for specifying user-defined data types. For instance, consider that we want to declare an element, which is to represent a month of the year. In DTD language the most we can say about such element is that it should be a string of characters.

```
<!ELEMENT MONTH #PCDATA>
```

The following element instance would be valid.

```
<MONTH> Monday</MONTH>
```

Of course it is always possible to think of some appropriate encoding which would represent better the intended meaning of such elements, but this implies complex definitions and moreover the way such encoding is to be interpreted should be remembered or implemented in ad hoc program applications. A third problem with the DTD language concerns reusability. Reusability in DTDs is done only through a macro-like mechanism using parameter entities. That means that the syntactical structure of such entities should be known and when reusing them, one should not use the same syntactical names as in the parameter entities. The worst thing about the DTD extension mechanism (parameter entities) is that it doesn't really make relationships explicit. Two elements defined to have the same content models aren't the same thing in any explicit way. Likewise, a group of attributes defined as a parameter entity and reused aren't logically a group, they're just "coincidentally" a group. And finally, the DTD language cannot be combined with namespaces in a trivial way. Thus it is not suitable for use in distributed environments.

In short, the DTD language is too much syntactical and tailored mainly to document rendition semantics. The XML Schema language tries to solve these problems. The additional expressiveness will allow web applications to exchange XML data much more robustly without relying on ad hoc validation tools.

XML Schema documents are XML documents. This means that they use elements and attributes to express the semantics of the schema and that they can be edited and processed with the same tools used to process other XML documents. The vocabulary of an XML Schema document is comprised of about thirty elements and attributes. In a somewhat recursive manner, XML Schema documents are valid only if they conform to the schema for XML Schema. There is also a DTD for XML Schema, see [Schema].

4.3 Style Sheets

The Extensible Style Sheet Language (XSL) is designed to apply representation style to XML documents. XSL is a specification under development with the World Wide Web Consortium for applying formatting to XML documents in a standard way. XSL includes both a transformation language (XSLT) and a formatting language. Each of these is an XML application. The transformation language has been recently standardized (recommendation of the W3C XSLT 1.0 November, 1999;). It provides elements that define rules for how one XML document is transformed into another XML document. The transformed XML document may use the markup and DTD of the original document or it may use a completely different set of tags. In particular, it may use the tags defined by the second part of XSL, the formatting objects. Examples of formatting objects are tables, horizontal and vertical rules, graphical objects etc. Formatting objects are defined as XML documents. Every formatting object is described in terms of the class it belongs to and these classes are defined as DTDs or schemes. The definition of formatting objects is not stable: many of them are not defined and not implemented and the implementation of others changes over time.

The transformation and formatting parts of XSL can function independently of each other. For instance the transformation language can transform an XML document into a well formed HTML file, and completely ignore XSL formatting objects. On the other hand, a document written in XSL formatting objects may not be necessarily produced by using the transformational part of XSL on another XML document.

4.3.1 XSL Transformations

The transformation language (XSLT) is not only used for document rendition applications. Its ability to transform data from one XML representation into another XML representation has numerous applications in XML-based electronic data interchange, metadata exchange and in general everywhere where there is a need to convert between two or more data representations.

In an XSL transformation an XSL processor reads an XML document and an XSL style sheet associated with it. Based on the instructions in the style sheet the XSL processor outputs another XML document or part of an XML document. There is special support for outputting HTML documents. The XSL processor reads in fact a parsed XML document (represented as a tree) and transforms it into another XML-tree. It can also transform an XML tree into a plain text, several elements or a mixture of elements and text.

Consider the following example.

```
<?xml version = "1.0" ?>
<?xml-stylesheet type="text/xsl" href = "partyInfo.xsl" ?>

<PARTYINFO party_type = "INSURANT">
       <NAME> Bob Jonson </NAME>
       <ADDRESS>
              <STREET> Viaweg </STREET>
              <NUMBER> 15 </NUMBER>
              <CITY> Enschede </CITY>
       </ADDRESS>
</PARTYINFO>
```

Example 1. An XML document

This document can be represented as a tree in the following way.

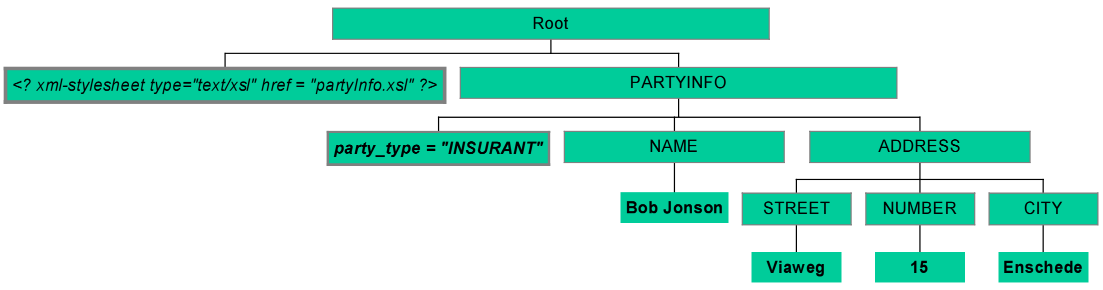


Figure 4-1. A tree representation of the PARTYINFO document

The XSLT language operates by transforming one XML tree into another XML tree. It contains operators for selecting particular nodes from the tree, reordering the nodes, and outputting nodes.

An XSL document contains a list of templates and other rules. A template rule has a pattern specifying the trees it applies to and a template to be output when the pattern is matched. When an XSL processor formats an XML document using an XSL style sheet, it scans the XML document tree looking through each sub-tree in turn. As each tree in the XML document is read, the processor compares it with the pattern of each template rule in the style sheet. When the processor finds a tree that matches a template rule's pattern, it outputs the rule's template. This template generally includes some markup, new data, and some data copied out of the tree from the original XML tree.

The XSL document itself is an XML document with a root element `<xsl:stylesheet>` or `<xsl:transform>`. Each template rule is an `<xsl:template>` element. An example of a stylesheet document, which is used to transform the XML document from Example 1 is shown below.

```
<?xml version='1.0'?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
      <html>
            <head>
         <title>
              Information about
              <xsl:value-of select="PARTYINFO/@party_type" />
          </title>
             </head>
             <xsl:apply-templates select=”PARTYINFO” />
       </html>
</xsl:template>
<xsl:template match="PARTYINFO">
        <body>
       <p>
            <xsl:value-of select="NAME" />
       </p>
       <p>
             Address:
              <xsl:value-of select="ADDRESS/STREET" />
       <xsl:value-of select="ADDRESS/NUMBER" />
              <xsl:value-of select="ADDRESS/CITY" />
       </p>
         </body>
</xsl:template>
</xsl:stylesheet>
```

Example 2. An XSL document

The XLS document of Example 2 consists of two template rules. The first one transforms the tree starting at the root node (see Figure 4-1) into a tree which is depicted in Figure 4-2.

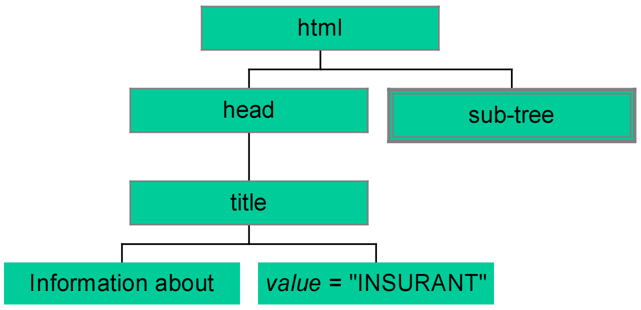


Figure 4-2. Template rule application: first template

The first template rule contains a rule `<xsl:apply-templates select="PARTYINFO" />`, which in fact results in applying the second template rule to the XML trees rooted at the node `PARTYINFO` (see Figure 4-1). The final tree obtained after applying the second template rule is the following.

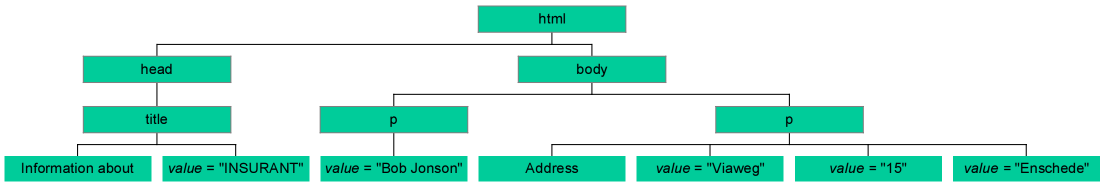
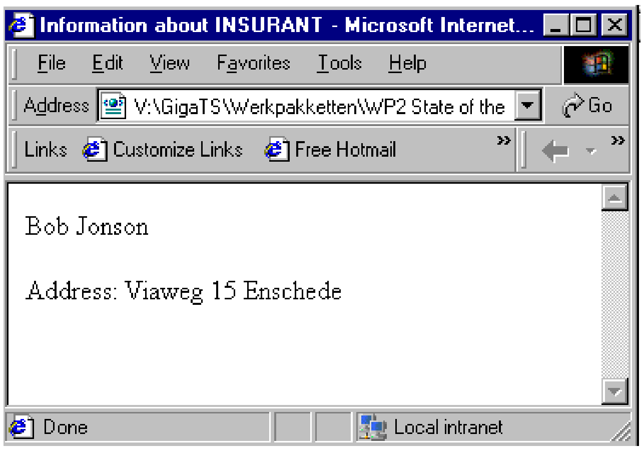


Figure 4-3. Template rule application: second template

The result of applying the style sheet to the XML document in Example 1 looks as follows.



Information about INSURANT - Microsoft Internet...
File
Edit
View
Favorites
Tools
Help
Address
V:\GigaTS\Werkpakketten\WP2 State of the
Go
Links
Customize Links
Free Hotmail
Done
Local intranet

Bob Jonson

Address: Viaweg 15 Enschede

5 XML tool support

This section aims to provide an overview of tool support for XML. We elaborate on existing XML processors and their functionality, XSL engines, XML browsers, XML editors, publishing systems, and finally database systems. This is the overview of October 2000, and like XML also its tool support is continuously extending. Hence, for an up-to-date overview of available tools we refer to http://www.xmlsoftware.com/, which provides well-organized information and resources on XML.

5.1 XML processors

The basic steps necessary for processing XML documents are

- lexical analysis
- syntactical analysis
- semantical analysis

The purpose of the lexical analysis in XML is recognizing the valid parts (words) of an XML document. These can be start and end tags, character data, processing instruction tags, etc. The lexical analysis is the basis of the syntactical analysis. The syntactical analysis is used in order to recognize valid language constructions in XML. For example, a start element tag should be followed by an end element tag. The syntactical analysis (also called parsing) is hierarchical while the lexical (also called scanning) is linear. The output of the scanning is a sequence of tokens while the parsing process groups these tokens into a tree structure. The semantical analysis checks the parse tree (produced by the syntax analysis) for semantic errors. In XML, semantic analysis is called validation and basically it consists of checking whether an XML document instance conforms to the DTD or Schema if such has been declared.

5.1.1 Lexical analysis

The lexical analyzer is usually not a standalone application. Its functionality is used by other applications. As mentioned before a parser uses a lexical analyzer in order to construct syntax trees. Sometimes lexical analysis is used directly by end applications without making a syntactical analysis. For example, consider a program, which finds in a collection of XML documents those which contain an element with name “ADDRESS”. This program can make use of a simple lexical analyzer recognizing just the start tags of elements. Upon finding a start tag with name “ADDRESS” in an XML document the lexical analyzer will give the control to the program for further processing of the document.

Lexical analyzers which work in the way described above are called event-based. The encountering for example of a start tag is considered as an event and the control is given to some external procedure capable of processing the event.

There is a popular event-based API developed by XML processor users and developers in an open discussion group called XML-DEV. The name of this API is SAX (standing for Simple API for XML). It comprises definitions of events, such as “start document”, “start element", "end element", "start attribute", etc. SAX gives a precise definition of the interface used to access functionality, which handles the corresponding events.

To understand how an event-based analyzer works, consider the following sample document:

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

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

1. start document
2. start element: doc
3. start element: para
4. characters: Hello, world!
5. end element: para
6. end element: doc
7. end document

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

5.1.2 Syntactical analysis

The application that performs the syntactical analysis is called parser. An XML parser uses the output of an XML lexical analyzer in order to produce a parse tree. The World Wide Web Consortium has standardized the way XML documents are represented as parse trees and the way these parse trees are accessed by other applications. The standard representation of an XML parse tree and the interfaces to it are called Document Object Model, Level 1 (DOM). The DOM Level 1 specification is a recommendation of the W3C since October 1998.

In general, DOM defines a tree-like representation of XML documents, but it does not put the restriction that documents must be implemented as trees. It also does not define how relations between documents are to be implemented. DOM is only the logical structure of the documents which is the interface to the actual implementation. How the parse tree is implemented is a decision of the parser vendors. The name "Document Object Model" was chosen because it is an "object model" in the traditional object oriented design sense: documents are modeled using objects, and the model encompasses not only the structure of a document, but also the behavior of a document and the objects of which it is composed. In other words, the nodes in the parse tree do not represent a data structure, they represent objects, which have functions and identity. As an object model, the DOM identifies:

- the interfaces and objects used to represent and manipulate a document
- the semantics of these interfaces and objects - including both behavior and attributes
- the relationships and collaborations among these interfaces and objects

The DOM is designed to be used with any programming language. In order to provide a precise, language-independent specification of the DOM interfaces, W3C has chosen to define the specifications in OMG IDL, as defined in the CORBA 2.2 specification (see [OMG]).

Figure 5-1 shows how applications can access the result of the parsing via DOM Level1. XML parsers compliant to DOM should provide implementation of the logical objects and relations specified in DOM. The application that uses the parser “sees” only the logical representation of the result of the parser. In this way application programmers can change the parser component in their applications.

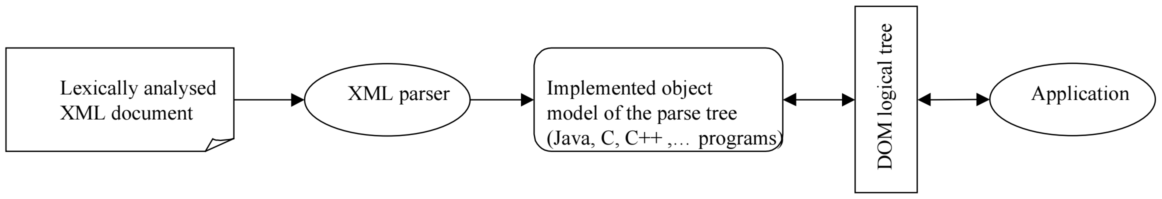

Lexically analysed XML document
XML parser
Implemented object model of the parse tree (Java, C, C++ ,… programs)
DOM logical tree
Application

Figure 5-1. XML parsing using DOM

5.1.3 Semantical analysis

Some applications are designed to process only XML files which conform to a DTD or a schema. For such applications not only the structure of an XML document is relevant but also the validity of the structure. The application, which checks whether the output of a parser conforms to a DTD or an XML Schema is called XML DTD or Schema validator. XML validators have always as central sub-functionality an XML parser, and for that reason they are often referred to as validating parsers.

An XML validator is very similar to a type checker in compilers (see [Aho86]). In general it augments the parse tree with some DTD or type information according to a given DTD or XML schema to which the XML file under processing should conform. The validator filters out parse trees for which this is not possible.

The W3C has tried to standardize also the logical structure of a validated tree. This is done in DOM Level 2 specification, which is a candidate recommendation since May 2000. DOM Level 2 is an extension of DOM level 1, which is capable to represent such semantically correct parse trees.

The validation process and the way DOM Level 2 objects are used is presented in Figure 5-2. The validator extends the implementation of syntax trees with semantical information about the types of node objects in the syntax tree. For example, if the implementation of the syntax tree is object-oriented, the extension can be done through object-oriented methods for extending classes.

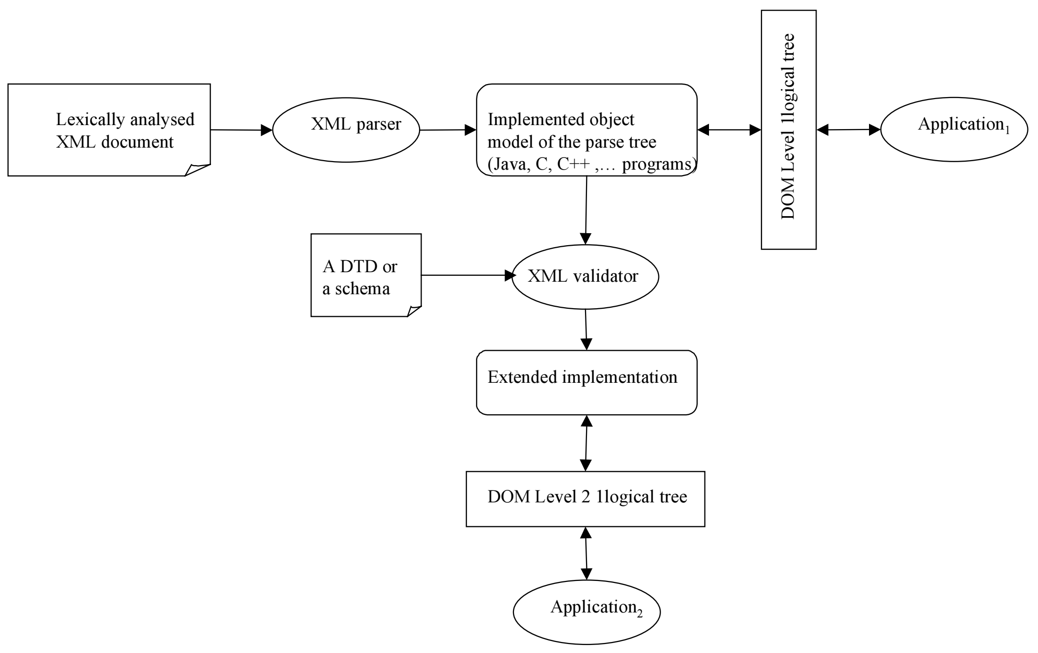

Lexically analysed XML document
XML parser
Implemented object model of the parse tree (Java, C, C++ ,… programs)
DOM Level 1logical tree
Application1
A DTD or a schema
XML validator
Extended implementation
DOM Level 2 1logical tree
Application2

Figure 5-2. XML validation using DOM level 2

5.1.4 Examples

In this section we list some examples of analyzing/parsing tools for XML documents.

- **expat** is a library, written in C, for parsing XML documents. It's the underlying XML parser for the open source Mozilla project, perl's XML parser, and other open-source XML parsers. It's very fast and also sets a high standard for reliability, robustness and correctness.
- **MSXML** Parser Beta Release, September 2000 (http://msdn.microsoft.com/downloads/webtechnology/xml/msxml.asp)
- **Python** (http://www.pythonlabs.com/products/python2.0/)
- **XML4C** (http://www.alphaworks.ibm.com/tech/xml4c) XML4C is a validating XML parser written in a portable subset of C++. A single shared library provides classes for parsing, generating, manipulating, and validating XML documents. XML4C is faithful to the XML 1.0 Recommendation and associated standards (DOM 1.0, SAX 1.0).
- **Xerces C++** (http://xml.apache.org/xerces-c/index.html) is a validating parser written in a portable subset of C++ that conforms to DOM 1.0 and SAX 1.0. Additionally, early implementations are included for DOM Level 2 and SAX version 2.0. Source code, samples and API documentation are bundled with the download.

5.2 XSL engines

In this section we will consider specific XML applications which make use of the basic XML tool support. A standard XML application is the XSL processor. A XSL processor transforms a document from an abstraction to a rendition. It works in two steps. First it transforms an XML document into a formatting object element and then it uses a predefined formatting objects vocabulary to represent these formatting objects. The representation can be displayed on the screen or may be converted to other format such as PDF, PostScript or RTF.

- XSL transformers: converts parsed XML documents into parsed XML documents. The principle way a XSL transformer works has been explained in section 4.3.1. A XSL transformer can be implemented at the client (browser) or server side. Some examples of XSL transformers are the following.
  - LotusXSL 1.0.1. implements XSLT transformations at the client and the server side. It is part of the IBM alpha works (see [IBMalpha]). LotusXSL can be interfaced with any parser that conforms to DOM Level 2 or SAX specifications.
  - Xalan is a XSL transformer, which is a part of the Apache's XML project. It can be interfaced with DOM and SAX parsers. In addition it supports its own input/output format which optimizes the DOM tree representation.
  - MSXML is the Microsoft implementation of an XML parser and an XSL transformer tool.
- XSL formatters. The XSL formatters translate XML documents presented as XSL formatting objects into existing formats or render them directly. At the moment there are not many implementations of XSL formatters since the specification of XSL formatting objects is not yet complete and standardized. As an example consider the FOP formatter which is a part of the Apache's XML project (see [Apache]).

5.3 XML browsers

We mention the most important XML browsers:

- **MS internet explorer** (http://www.microsoft.com/windows/ie) uses the MSXML parser.
- **Netscape Navigator** (http://www.netscape.com/browsers/6) uses the expat parser. Netscape 6 fully supports HTML 4.0, XML, CSS1, DOM1, and Resource Description Framework (RDF). It also supports XML Namespaces and has support for CSS2 and DOM2.
- **Opera** (http://www.opera.com/) has implemented XML internally, and this will be supported in version 4.0.

5.4 XML editors

Strictly speaking, an XML editor is a standalone software product for creation and editing of XML documents. But since XML documents can be edited in the most simple text editor, XML editors usually offer a much broader range of possibilities, including intelligent editing of XML documents, schemata and style sheets, but also XML import from (and export to) databases, immediate wellformedness checks and validation of XML documents, schema generation and conversion, XSL transformation, and much more. In other words, most XML editors are complete integrated software packages. In this section, we offer a small survey of core functionality of XML editors: XML editing, schema editing, and style sheet editing.

5.4.1 XML editing

Although XML editing can be done in a simple text editor, most XML editors offer much more sophisticated editing such that using an XML editor is really worthwhile. Most editors assist the user in immediately producing a well-formed and valid XML document. To this end, they offer the use of templates and intelligent editing. Given a particular document type definition or schema the editor creates a template containing the link to the appropriate DTD or schema, as well as the corresponding root element ready to be filled in. Intelligent editing assists the user by –for example- different coloring of syntax, providing the end tag when the start tag has been typed, providing a list of available items (elements, attributes and values) that can be inserted in a certain location according to the schema associated with the XML document, or by providing all sub-elements for a root element. Finally, an editor may offer immediate wellformedness checks and validation of the XML documents using the (external) XML processor of the user’s choice.

5.4.2 Schema editing

Basically, three different ways exist to create an XML schema:

1. Generation: create a schema design given the requirements for the document structure.
2. Reverse engineering: given an existing XML document deliver the schema which describes the structure of the XML document most precisely.
3. Conversion: given an existing DTD, convert all structure information to a schema

Most editors support these three ways of schema creation. Additional functionality might include schema validation (check the schema or DTD for compliance with the corresponding rules for creating such a schema or DTD), intelligent schema editing, and support of different schema languages (DTD, XDR, BizTalk, XML schema).

5.4.3 Style sheet editing

Again, most editors offer intelligent style sheet editing. Additionally, the user can often select his preferred XSL transformation engine to transform the XML document to another XML document or to other formats like (HTML, pdf, postscript, et cetera).

5.4.4 Examples

Today, too many XML editors exist to treat them all in this section. Therefore, we selected a few of the editors with most functionality.

- **Xeena** (http://www.alphaworks.ibm.com/tech/xeena) XML Editing Environment, naturally in Java. It can take any DTD, and lets the user create and edit XML files that conform to it.
- **XmetaL** (http://www.softquad.com/products/xmetal) is an XML and SGML authoring tool that creates documents that conform to arbitrary DTDs. It features three editing views (normal, plain text, tags on), Advanced Authoring Aids (show valid markup options at the current point in the document, drag and drop), a resource manager and database import.
- **XML Instance** (http://www.extensibility.com/products/xml_instance/index.htm) enables users to edit and validate XML business documents (i.e. instance documents) that conform to a DTD or XML schema in any prominent XML schema dialect. Data from an ODBC data source can be imported deployed throughout the organization as XML data – offering a means to integrate and reuse legacy data.

- **XML Spy** (http://www.xmlspy.com) is a validating XML editor that provides three integrated views on XML documents: an enhanced grid view, a low-level text view with syntax coloring, and an integrated browser view that supports CSS and XSL style-sheets. It is a very professional editor with detailed find, replace, and print options that are available in all views. Complete Unicode and character-set encoding support is included. Has project management to organize the collection of XML, XSL and schema documents and can perform batch operations.
- **XML Writer** (http://www.xmlwriter.net) provides users with an extensive range of XML functionality such as: validation of XML documents against a DTD or XML Schema, and the ability to convert XML to HTML using XSL stylesheets. Users can also combine CSS with XML for direct formatting of XML data. Especially designed to manage projects, XMLwriter's Integrated Development Environment (IDE) provides useful project management features such as a structured Project View workspace, and batch validation of files within a project.

5.5 Database systems

An extensive overview of XML databases is given by Ronald Bourret. He distinguishes between the following categories of products [Bourret]

1. Middleware: Software you call from your application to transfer data between XML documents and databases.
2. XML-Enabled Databases: Databases with extensions for transferring data between XML documents and themselves.
3. XML Servers: Platforms that serve data -- in the form of XML documents -- to and from distributed applications, such as e-commerce and business-to-business applications.
4. XML-Enabled Web Servers: Web servers that serve XML -- usually built from dynamic Web pages -- to browsers.
5. Content Management Systems: Systems for managing fragments of human-readable documents and include support for editing, version control, and building new documents from existing fragments.
6. Persistent DOM Implementations: DOM implementations that use a database for speed and to avoid memory limits. These usually also support an XML query language.

We shortly list a number of important XML database products

- **Excelon** (http://www.exceloncorp.com/products) An XML data server for building enterprise web applications. Allows the storage and management of native XML. XML is parsed and then stored in its parsed format. Supports XML, XSL, SQL, XQL, DOM, Java, and COM. Nice features include: eXcelon Explorer, a browser to view, import, organize, modify, query, and set securities on data visually, a WYSIWIG tool for creating XSL stylesheets, and eXcelon Studio, to define XML schemas and generate XML-based and data-driven web pages for rapid application development.
- **Frontier** (http://frontier.userland.com) is an integrated Internet content system built around key features such as a high performance object database, outliner, scripting, and a multi-threaded runtime.
- **ODBC2XML** (http://www.xmlmethods.com) Merges data from any ODBC data source into XML documents. It accepts well-formed XML "template" files that can contain XML text, SQL queries and instructions about where the query results merge with the XML.

- **Cocoon** (http://xml.apache.org/cocoon) Cocoon is an XML publishing framework for the Apache web server. Web pages are written in XSP (eXtensible Server Pages), a scripting language. This language retrieves data from the database with the SQLProcessor module, which uses JDBC. Web pages that need data include embedded SQL statements, which can be parameterized. An overridable method can be used to create even more complex SQL statements. SQLProcessor returns the results as XML. It models the XML as a table, using the same DTD as the Oracle XSQL Servlet. The user can specify the element names to be used at the table and row level. Column-level element names are taken from the database.
- **Target 2000** (http://www.target2000.com/main.html).

6 XML in finance

XML will play an important role in the financial sector. Many different initiatives exist. Here we consider the Open Financial Exchange, Interactive Financial Exchange, and Financial XML protocols.

6.1 Open Financial Exchange

Open Financial Exchange (OFX) is an initiative of CheckFree, Intuit, and Microsoft [OFX]. CheckFree Corporation [CheckFree], is a provider of financial electronic commerce services and products in USA mainly. Intuit provides services and software for personal finance, small business accounting, and tax preparation. Intuit is headquartered in Mountain View, California, and employ approximately 3,500 people in the United States, Asia, and Europe.

OFX supports a broad range of financial services by defining a framework for directly exchanging financial data and instructions between customers and their financial institutions. Customers can be individual consumers, tax payers, small and middle businesses. Financial institutions can be all organizations providing some kind of financial services, thus including banks, financial advisors, government agencies, merchants, businesses, information providers, transaction processors, brokerage houses, etc.

For each financial service, OFX defines the format of financial data exchanged, the set of instructions (messages) and the way the data and the instructions are interpreted. The financial data and instructions are specified in SGML. The interpretation of data and instructions is realized by implementing OFX server and client applications.

The financial services supported by OFX are classified into four groups, namely pure banking services, bill presentment, payment, and investment services.

- **banking services**: Banking services are the services regarding manipulations of customers' accounts. OFX supports both customer and business banking. The following banking services are considered by OFX.
  - *Statement download* (downloading transactions and balances). Bank and credit card statement downloads are supported.
  - *Stop check* (stopping payment for one or more checks).
  - *Funds transfers* (transferring a specified amount from an account to another). OFX supports intrabank, interbank, and wire funds transfer.
  - *E-mail and customer notification.* OFX enables the clients to contact their Financial Institutions when they have questions regarding their accounts. Financial Institutions can notify their clients for certain events (such as returned check and returned deposit) as well.

- **bill presentment.** OFX specifies how to deliver electronically bills from a biller to a customer. The services related to bill presentment are finding billers and bill publishers, enrolling with bill publishers, requesting bills for one or more accounts from one or more billers and paying the bills (via integration with OFX payment services).

- **payment services:** OFX supports consumer and business payments. For business payments there are additional features, such as distribution of payments over multiple invoices. Payments of bills and invoices can be done using bank or credit card accounts. OFX distinguishes between two types of payees. “Standard payees” are those which receive recurring payments from multiple users. “Any” payees are those for which single or recurring payments are scheduled from single users. For each type of payment OFX defines the following operations:
  - payment creation
  - payment modification
  - payment status inquire
  - payment cancellation

  In addition the clients can inquire their Financial Institutions or payment processors via e-mail.

- **investment services.** OFX supports download of securities information and etailed investment account statements including transactions, open orders, balances and positions.

OFX relays on supporting services, such as Trusted Third Party Services (TTP), security services and communication services in order to provide the financial services above.

The OFX specification is publicly available on the Internet and everybody can implement OFX clients and servers. The specification uses accepted standards for:

- data specification: SGML and XML are used to specify the format of request and response messages, as well as the format of financial data interchanged between the OXF clients and servers.
- communication: the communication is based on HTTP, and thus can be carried over the Internet.
- security: transport level security uses SSL to secure communications between OFX servers and clients.

The OFX specifies a client-server architecture. The server implements financial services or has access to such, and the client can request such services. The communication between the client and server is carried out in HTTP (see Figure 6-1).

The OFX client issues OFX-requests, which are translated into HTTP requests (using the HTTP Post command) and sent to a web-server. The web-server transforms the requests into OFX requests and forwards them to a OFX server. The OFX server performs the services requested or further forwards the requests to other server applications (such as a legacy system) for processing. After the service requested is performed, the OFX server produces a replay and sends it via the web-server and the web-client to the OFX client.

The client can send requests to the OFX-server only after locating the corresponding web server and obtaining information on which services are supported by that particular OFX server. For that purpose, each Financial Institution supports a Profile-server, which contains location information for the web-servers and the sets of services supported by each of the OFX servers connected to the web-servers (see Figure 6-2).

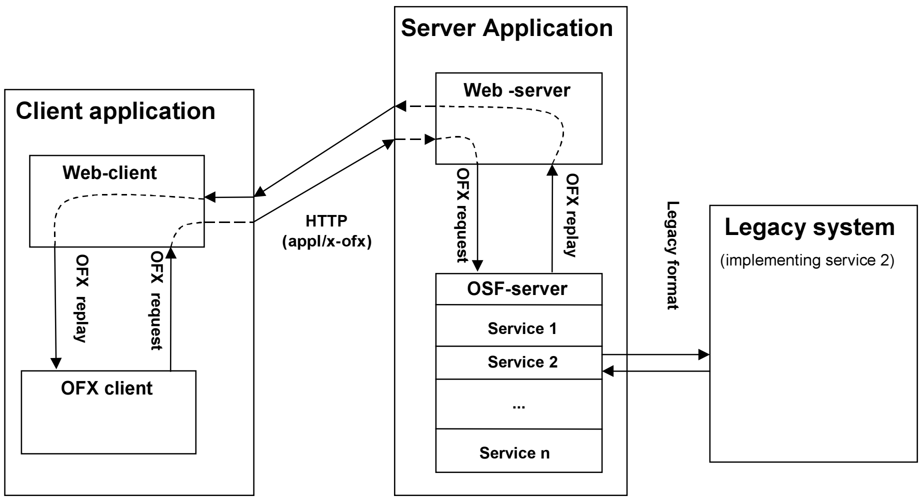


Figure 6-1. OFX architecture

OFX clients and servers use transport level SSL to secure communications. In addition, OFX servers are required to have a digital certificate to authenticate the server representing the financial institution or the service provider. The OFX group working with VeriSign has developed a separate Certification Authority (CA), the OFX CA, to issue OFX certificates. The OFX CA is VeriSign. It incorporates leading-edge implementation of proven security technology and features a range of security solutions, including transport level SSLv3 and application-level security using secure, sealed envelopes.

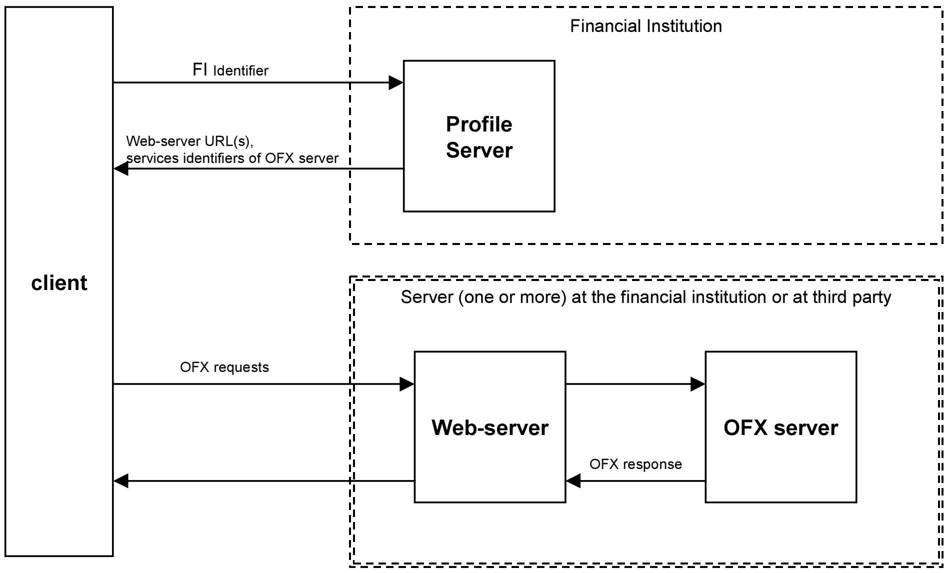

Financial Institution
FI Identifier
Profile Server
Web-server URL(s), services identifiers of OFX server
client
Server (one or more) at the financial institution or at third party
OFX requests
Web-server
OFX server
OFX response

Figure 6-2. OFX architecture

6.2 Interactive Financial Exchange

The Interactive Financial Exchange (IFX) is an initiative of the IFX forum [IFX]. The IFX Specification (V1.0.1 as of 04/04/00) provides a robust and scalable framework for the exchange of financial data and instructions independent of a particular network technology or computing platform. It was developed as a co-operative industry effort among major financial institutions, service providers and information technology vendors serving these institutions and customers in the small business and consumer markets. IFX builds on the industry experience of the OFX Specification. (see section 6.1). At the IFX Forum's web site we find:

> *"IFX 1.0 was based on the "InteroperaBILL" initiative. Financial institutions, technology providers and billers participated in the effort to develop business level technical requirements to build an interoperable, online bill presentment and payment solution. As part of the "InteroperaBILL" initiative, the National Automated Clearing house Association's Council on Electronic Billing and Payment, created and published business guidelines for the developing online bill presentment and payment market."*

Electronic bill delivery and payment business messages are highlighted in the IFX Specification. These messages were developed to support the open, interoperable and secure exchange of electronic bills and the reliable delivery of payment and remittance information.

6.3 Financial XML

FinXML (www.finxml.org) is an XML based framework developed to support a single universal standard for data interchange within the Capital Markets. As such, FinXML can be used as the basis for straight through processing (STP) and risk management within a financial institution as well as conducting e-commerce over the Internet.

The FinXML consortium is responsible for the definition and dissemination of the FinXML specification and related vocabularies. The consortium is made up from financial institutions, technology vendors, systems integrators and others involved in the Capital Markets. FinXML is a framework within which vocabularies for capital markets can be defined. It also provides a framework within which applications that use these vocabularies can be developed and deployed. As such, it represents the underpinnings of a full solution for electronic commerce within an institution and between banks and their customers. Current vocabularies defined as part of FinXML support a broad set of elements and attributes that represent financial transactions, reference data, market data, payments, settlements and confirmations. FinXML supports a wide variety of financial products including interest rate, foreign exchange and commodity derivatives, bonds, money markets, loans and deposits, and exchange traded futures and options.

FinXML claims to be interoperable with existing financial protocols such as FIX and S.W.I.F.T. In addition, FinXML is compatible with outside industry standards such as BizTalk and cXML, which support e-commerce and EDI. The FinXML specification and DTDs for FinXML vocabularies are currently available to consortium members, however, this information will soon be made generally available. FinXML is defined by the FinXML consortium. At the moment version 1.0 is available. A key player seems to be the company Integral.com.

In short, FinXML provides XML-representations of cash flows, settlement information for interest rate and currency derivatives, foreign exchange, loans and deposits, interest rate futures and options, etc.

7 XML in trading

Several organizations and companies are now working on the building blocks of e-commerce, especially in the b2b setting. We here discuss a number of developments that are based on XML, being the Internet Open Trading Protocol, Common Business Language, Commerce XML, Open Buying on the Internet, XML/EDI, and Trading Partners Agreement Markup Language.

7.1 Internet Open Trading Protocol

The IOTP is a framework for Internet commerce, developed by the TRADE working group of the IETF, on the initiative of the OTP Consortium. The consortium consists of a number of important players on the Internet payment market and some major hardware companies. The framework aims to offer capabilities to replicate the real trade world in the virtual world of the Internet. The protocol was submitted as Internet Draft [IOTP]. It now is an international Request for Comments [RFC IOTP].

The IOTP offers Internet versions of accepted methods for trading, selling, buying, and value exchange. Furthermore, it must serve for the introduction of new trading models. The IOTP wants to define these trading events in such a way that confirmation to the IOTP specifications guarantees safe and successful completion of transactions.

The most common ways of doing trades on the Internet are specified in Baseline IOTP. The approach taken is to give limited, but usable specifications and let pilot products be developed and placed in the market to understand real demands and requirements. The Internet Draft [IOTP] serves as the specification, and may be used by developers of e-commerce products to base their products on IOTP. The IOTP focuses on how e-commerce products communicate: it describes content, format and sequences of messages sent between the participants in an electronic trade. It does not prescribe the software and processes for each participant.

The following types of transactions are defined:

- purchase
- refund
- value exchange
- authentication
- withdrawal
- deposit
- payment instrument customer care
- inquiry
- ping.

A transaction involves a number of organizations playing a trading role. The protocol distinguishes the trading roles customer, merchant, merchant customer care provider, payment handler, deliverer, and payment instrument customer care provider.

The interaction between trading roles can be characterized by four patterns, called trading exchanges: offer, payment, delivery, and authentication. Each transaction is composed of a number of trading exchanges. For example, a purchase is the composition of offer, payment, and delivery. A trading exchange is built from pre-defined trading components. In a trading exchange these components are transmitted between the trading roles. For performance reasons trading components for several trading exchanges are packed together into one message during transmission. In the same way trading exchanges are intermingled in transactions.

The framework is payment scheme independent and encapsulates several schemes, such as SET, Mondex, CyberCash, and DigiCash. Each payment scheme contains message flows that are specific to the scheme; these parts of the protocol are contained in a set of payment scheme supplements.

7.2 Common Business Language and Commerce XML

Commerce One's Common Business Library (xCBL) 2.0 is the first open XML specification for the cross-industry exchange of business documents such as product descriptions, purchase orders, invoices, and shipping schedules. xCBL 2.0 is a set of XML building blocks and a document framework that allows the creation of robust, reusable, XML documents for electronic commerce. Using the xCBL 2.0 document framework, businesses everywhere can exchange business documents of different types, resulting in frictionless electronic commerce across multiple trading communities. For businesses already using traditional Electronic Data Interchange (EDI) standards, xCBL 2.0 provides a transition path to an XML-based commerce capability.

xCBL 2.0 is the first XML specification for electronic commerce designed to take advantage of the expressive power of XML schemas. xCBL is available in XML DTD form and in two different schema languages: Microsoft's XML Data Reduced (XDR) and Commerce One's Schema for Object-oriented XML (SOX). The upcoming W3C Schema Definition Language will also be supported.

xCBL 2.0 is not a single standard; rather it is a collection of common business elements that underlie all EDI and Internet commerce protocols. xCBL 2.0 has been developed and modeled after EDI semantics such as X12 and EDIFACT to preserve and extend the EDI investments of the trading partners. Its reusable components speed the implementation of standards and facilitate their interoperation by providing a common semantic framework. xCBL 2.0 is developed by CommerceOne. xCBL has the support of Microsoft's BizTalk effort.

Arriba is another company that has a strong position in b2b e-commerce. They have proposed cXML (cxml.org), which is quite similar to CBL. Here they also define messages for the usual data exchanges in a b2b e-commerce setting, plus that they define a protocol to exchange them on top of that.

There seems to be some rivalry between the two standards. cXML is incorporated in the BizTalk framework (Microsoft), whereas CBL is adopted by Commerce Net within the eCo Interoperability framework. It is not clear what direction this will go.

7.3 Open Buying on the Internet

The OBI (Open Buying on the Internet) standard is an open, freely available framework for high volume, low dollar, business-to-business transactions. The OBI consortium [OBI] claims that these transactions form 80 % of all transactions. The standard consists of an architecture, technical specifications, and guidelines [OBI98]. The standard is vendor neutral. Out of practicality, it is based only on technologies that were available in 1997.

The standard was defined by the Internet Purchasing Roundtable, a forum consisting of important companies and suppliers of indirect materials. After definition of the standard the OBI consortium was formed to support the standard. The consortium consists of buying and selling organizations, technology companies, financial institutions, and others. The goal of the consortium is to promulgate and enhance the standard, and to develop new, other standards for Internet commerce.

The basic principle of the OBI architecture is that process owners are responsible for all information associated with their business processes. Buying organizations, for example, are responsible to keep their own requisitioner profile up to date; this contrasts with consumer Internet commerce, where the seller collects and stores this data. The architecture uses a model of business-to-business commerce, which is explained below.

1) A requisitioner, using a Web browser, connects to a local purchasing server located at the Buying Organization and selects a hyperlink to a Selling Organisation's merchant server containing an on-line catalogue of goods and services.

2) The Selling Organisation's server authenticates the requisitioner's identity and organisational affiliation based on information presented in the requisitioner's digital certificate. Authentication information is used, in conjunction with profile information optionally presented by the requisitioner's browser, to uniquely identify the requisitioner and to construct a specialised catalogue view. The requisitioner browses the catalogue, select items, and "checks out."

3) The content of the requisitioner's "shopping basket" and identity of the requisitioner is mapped into an order request (EDI-compatible). A digital signature is calculated (optionally); the order request (and digital signature if used) is encapsulated in an OBI object which is encoded and transmitted securely to the Buying Organization over the Internet using HTTP and SSL. In OBI/1.1 there are two alternative methods for transmitting an encoded OBI object containing an order request over the Internet using HTTP. These are referred to as the server-to-server method and the server-browser-server method. The Buying Organization server receives the encoded OBI object, decodes it, extracts the order request, verifies the signature (if appropriate) and translates the order request into an internal format for processing.

4) Administrative information (including payment type) is added to the order request at the Buying Organization (automatically from a profile database and/or manually by the Requisitioner), and the order is processed internally either automatically or through a workflow-based process.

5) The completed and approved order is formatted as an OBI order (EDI-compatible) and a digital signature is calculated if desired. The order (and digital signature if appropriate) is encapsulated in an OBI object which is encoded for transport and transmitted securely from Buying Organization server to Selling Organization server via the Internet using HTTP over SSL. The Selling Organization receives the encoded OBI object, decodes it, extracts the order, verifies the signature (if appropriate), and translates the order into its internal format.

6) The Selling Organization obtains credit authorization, if necessary, and begins order fulfillment.

7) The payment authority issues an invoice and receives payment.

The OBI architecture uses the SSL protocol on the Internet, x.509v3 for digital certificates, HTTP over SSL, and ANSI ASC X.12's 850 standard EDI purchase order.

OBI V1.1 was implemented in several successful pilot projects. V.2.0 moves the OBI specification from the testing phase into production implementation. In V2.0, OBI now supports multi-vendor requirements, customer-specific catalogues, and secure processing on the Web. Purchase orders fed directly into the customer's local procurement or finance system are returned to the seller via the Internet, eliminating the need for duplicate data entry by either the customer or the seller. Features include guidelines on using EDI-based applications with OBI-compliant trading partner systems, as well as standard processes for catalogue access, standard data formats for order-related information, standard methods for transmitting order-related data between organizations, and standard security mechanisms for authentication, secure communications, and non-repudiation. Additionally, OBI V2.0 supports multi-national transactions; allowing trading partners to manage procurement using various currencies.

7.4 XML/EDI

The explicit possibility to define and communicate about the structure of documents makes XML suitable for the exchange of structured data, like is the case in EDI. Currently therefore, work is underway in which EDI message formats are being defined in XML: XML/EDI, developed by the XML/EDI Group [XMLEDI].

This promises to bridge the often mentioned gap between Internet (human-computer-communication) and EDI (computer-to-computer communication) [Blom88, Hous95].

XML/EDI is a synthesis of many concepts. XML/EDI

- uses the XML protocol as its "data interchange modeling" layer
- uses the XSL protocol as its "presentation" layer
- can be integrated with traditional methods of Electronic Data Interchange (EDI)
- can be used with all standard Internet transport mechanisms such as IP routing, HTTP, FTP and SMTP
- allows for document-centric views and processing methodologies
- uses modern programming tools such as Java and ActiveX to allow data to be shared between programs

- uses agent technologies for data manipulation, parsing, mapping, searching.

XML messages not only contain data, but structure and format data as well. This structure and format data is included in so-called process templates and document type definitions. By reading the structure data the receiver of the message can interpret the message. XML/EDI also uses this principle, which enables partners to synchronise easily: with XML/EDI it is not necessary to engage in an agreement procedure before the start of the message exchange. EDI protocols are easily translated to XML/EDI. XML encapsulates data in tokenized formats and structures, the way EDI does. EDI segment identifiers are replaced by XML tokens.

A second advantage over traditional EDI is that so-called style sheets can be used to make the messages readable to humans. Traditional EDI is aimed at application-to-application communication and to allow a human to read or create a message a dummy application must be written. By using XML, a much larger audience can be reached: a user only requires a web browser and an Internet connection.

A third advantage is that the use of XML allows the embedding of EDI messages in the workflow of electronic commerce applications. Furthermore, where traditional EDI is inflexible, XML/EDI processes are extendible.

In conclusion, XML/EDI is especially suitable for short term trading relations and for SMEs. Initiatives in XML/EDI are emerging fast. Translations from EDIFACT to XML/EDI are written, all proprietary, for example in the XCAT project. Pilot projects are executed; an example is the EXPERTS project of ECP.NL, funded by the EU. At this moment there is no consistent generic foundation for XML/EDI initiatives. For example, no standard way exists to base messages upon a global repository like EDIFACT, but standards are being proposed now.

7.5 Trading Partners Agreement ML

Trading Partner Agreements ML, TpaML[1], is the result of one of IBM's research projects and is now brought within the context of ebXML (section 9.3). It is a framework for multi-party E-commerce. The aim is setting up and keeping distributed, long running business deals spanning multiple autonomous business organizations. An example is a package deal between a travel agent and an airline, hotel and car rental: if a customer purchases full fare on Airline B, he receives a 20% discount at Hotel C and a 10% discount at car Rental D.

The trading partner agreement holds an important position in the TpaML-concept because it's the template for all communications in a long-running transaction. It defines the properties of each party: which role they have, which position, which actions they're allowed to take etc.

The purpose of the electronic TPA is to express the IT terms and conditions to which the parties to the TPA must agree in a form in which configuration information and the

[1] http://www.ebxml.org/project_teams/trade_partner/trade_partner.htm

interaction rules which must be executable can be automatically generated from the TPA in each party's system.

It should be understood that the information in the TPA is not a complete description of the application but only a description of the interactions between the parties. The application must be designed and programmed in the usual manner. As a simple example, the TPA may define requests such as "reserve hotel". The "reserve hotel" function must be designed, coded, and installed on the hotel server. That function may, in turn, invoke various site-specific functions and back-end processes whose details are completely invisible to the other party to the TPA.

The overall XML structure of the TPA is as follows. Each of these tags is the top level of a subtree of tags (subelements):

```
<TPA>
<TPAInfo> <!-- TPA preamble -->
... <!--TPAname, role definitions, participants, etc.-->
</TPAInfo>
<Transport>
... <!--communication and transport security information-->
</Transport>
<DocExchange>
... <!--document-exchange and message security information-->
</Security>
<BusinessProtocol>
<ServiceInterface> <!-- for each provider-->
... <!--Action definitions etc.-->
</ServiceInterface>
</DocExchange>
</TPA>
```

8 XML for multimedia and metadata

XML becomes popular also in the area of representing multimedia data and metadata about it. Metadata describes the data in a human readable format and can be used for indexing and searching. There are various emerging standards, which have developed sets of metadata elements to facilitate the retrieval of multimedia resources. Many of them use XML, since XML is a metalanguage and that makes it an useful tool for describing metadata. XML based initiatives that are likely to develop into general standards are the Resource Description Framework and MPEG-7.

8.1 Resource Description Framework

The Resource Description Framework (RDF) is a basis for processing metadata and provides facilities to enable automatic processing of electronic resources. RDF is an approved recommendation of the W3C dating from February 1999. The document 'Resource Description Framework (RDF) Model and Syntax Specification' [RDF] introduces a model for representing RDF metadata. The basic data model is described by three object types: resources, properties, and statements. Each RDF expression is represented using this model. The syntax used for representing resources using the RDF data model is based on XML. RDF can also be represented by using triples or graphs. All representation formats have equivalent meaning and are not intended to constrain the internal representation within implementations.

A sentence like: "Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila" can be represented in RDF/XML as:

```
<?xml version="1.0"?>
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

This sentence is described using the triple:
`{creator, [http://www.w3.org/Home/Lassila], "Ora Lassila"}`

and can be represented using the following graph:

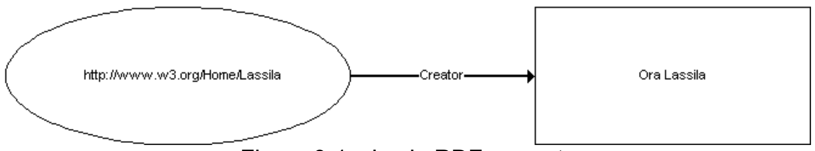


Figure 8-1: simple RDF property

RDF was developed to accommodate knowledge representation capabilities in the web. The fact that it has explicit unique identifiers in the form of URLs for the name spaces of metadata elements makes it suitable for multi-domain applications where, e.g., metadata produced by different communities has to be disambiguated. The World Wide Web Consortium, where RDF is standardized, proposes RDF as a technology to facilitate the 'semantic web' that incorporates inferential reasoning about resources, see [SemWeb].

8.2 MPEG-7

MPEG-7 is an ISO/IEC standard developed by MPEG (Moving Picture Experts Group), the committee that also developed the standards known as MPEG-1 and MPEG-2, and the 1999 MPEG-4 standard. While the other MPEG standards describe compression formats, MPEG-7 aims to create a standard for describing the multimedia content data that will support some degree of interpretation of the information's meaning, which can be passed onto, or accessed by, a device or a computer code.

MPEG-7 specifies a standard set of descriptors that can be used to describe various types of multimedia information. MPEG-7 will also standardize ways to define other descriptors as well as structures (Description Schemes) for the descriptors and their relationships. MPEG-7 is expected to be approved in June 2001.

MPEG-7 addresses applications in which video and audio content can be stored or streamed and can operate both in real-time and non real-time environments. Automatic feature extraction will not be within the scope of the MPEG-7 project, nor are search languages specified. The focus is on standard description elements of multimedia data.

The language that is standardized to specify Description Schemes and Descriptors in MPEG-7 is called the Description Definition Language (DDL). A hierarchical structure of Descriptors and Description Schemes is used to describe the multimedia data. The current draft set of metadata elements indicates a strong focus of MPEG-7 on contents features such as video frame attributes but also appears to have sufficient detail in the area of bibliographical metadata.

An MPEG-7 example of content description of a video is given below.

```
<ObjectDS id="object_chris_vissers">
  <AnnotationDS>
    <Who> Chris Vissers </Who>
  </AnnotationDS>
</ObjectDS>
<VideoSegmentDS id="clip_00.00.51_00.01.05">
  <TimeDS>
     <TimeD>
        <h> 00 </h>
        <m> 00 </m>
           <s> 51 </s>
        </TimeD>
        <DurationD>
           <h> 00 </h>
           <m> 00 </m>
           <s> 14 </s>
        </DurationD>
    </TimeDS>
</VideoSegmentDS>
<Link>
  <LinkSource id="object_chris_vissers" />
  <LinkTarget id="clip_00.10.51_00.11.45"/>
  <LinkTarget id="clip_00.16.24_00.18.07"/>
</Link>
```

Within the 'ObjectDS' tag given in the example above, the keyword 'Chris Vissers' is defined. Next, within the 'VideoSegmentDS' tags, one of the video fragments of a resource in which the person occurs is specified. The whole set of fragments in which the keyword 'Chris Vissers' occurs is then given within the 'Link' tags. Note that the 'LinkTarget' tags hold the identifiers of the resources that are described by the metadata in this fragment.

An authoritative source for MPEG developments is http://drogo.cselt.stet.it/mpeg/standards/mpeg-7/mpeg-7.htm.

9 XML frameworks

In interoperability and XML a number of different initiatives have started. First we discuss the eCo framework and BizTalk, two competing initiatives. Finally, an overview of a new initiative, the ebXML is presented. The goal of ebXML is to provide a common platform for conducting electronic business and to integrate frameworks as eCo and Biztalk.

9.1 eCo Framework

The goal of the eCo framework project, steered by CommerceNet, is to develop a specification that enables e-commerce system interoperability. eCo is intended as a specification within which industries and companies can innovate. eCo is not intended to compete with e-commerce transaction protocols or business semantic languages (BizTalk, CBL, XML EDI, RosettaNet, OBI, OFX, OTP, e"Speak, etc.)

Objectives of the eCo Framework are:

- to create a consistent yet extensible way to define and characterize online marketplaces from the most abstract level to the most detailed
- to express interoperability by coming at the problem from the perspective of business and business processes, so that businesses can better understand the issues of getting into an online marketplace, and what they must do yo solve them and prosper
- to make businesses interoperable to a minimum standard with only about 1 hour of investment—keeping the barriers to entry low for electronic commerce
- to create recommendations for XML businesses documents are machine readable, yet human-understandable

The eCo Interoperability Framework consists of two parts: the Architecture Specification and the Semantic Recommendations. The eCo Interoperability Framework is based overall on XML, which creates the possibility for automated buying and selling online. For instance, XML can be used to create computer-readable online catalogs and other business documents. The eCo Semantic recommendations document offers a set of recommendations for creating business documents in the form of Document Type Definitions (DTDs) and DTD Schemas so that those documents will be highly interoperable with other online businesses.

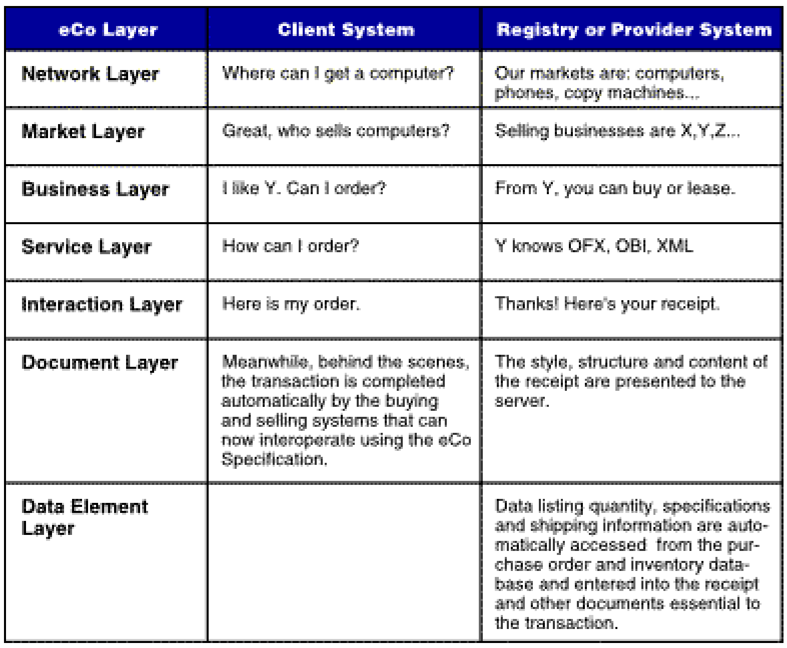
The eCo Architecture has a 7-layer structure that lets businesses describe an electronic marketplace from the highly abstract level of a network of marketplaces down to the level of individual protocols used to exchange business documents. It describes a set of interfaces that must be implemented to communicate at each of those levels. The architecture is extensible at each layer, including the capability to specify or define a set of transaction protocols.

Each layer of an eCo-compliant e-commerce system presents information about itself. By examining this information, others can:

- locate the system
- understand what it is for
- recognize what market(s) it participates in

- identify protocols the system uses to communicate
- discover what documents the system uses to conduct business
- learn how to interoperate with the system

For example, if a business wants to buy the next generation of laptop computers for its sales force, they can fill out a query form and launch an eCo-compliant intelligent agent to initiate the following transaction: (from http://eco.commerce.net/how/index.cfm)



eCo Layer	Client System	Registry or Provider System
Network Layer	Where can I get a computer?	Our markets are: computers, phones, copy machines...
Market Layer	Great, who sells computers?	Selling businesses are X,Y,Z...
Business Layer	I like Y. Can I order?	From Y, you can buy or lease.
Service Layer	How can I order?	Y knows OFX, OBI, XML
Interaction Layer	Here is my order.	Thanks! Here's your receipt.
Document Layer	Meanwhile, behind the scenes, the transaction is completed automatically by the buying and selling systems that can now interoperate using the eCo Specification.	The style, structure and content of the receipt are presented to the server.
Data Element Layer		Data listing quantity, specifications and shipping information are automatically accessed from the purchase order and inventory database and entered into the receipt and other documents essential to the transaction.

9.2 BizTalk

Microsoft has started Microsoft BizTalk (http://biztalk.org). There are two core issues behind the BizTalk initiative:

- Application integration today is too hard. The cost and complexity of integrating ERP systems, inventory management systems, or sales tracking systems within a single organization strains the capabilities of most large and small companies alike.
- The next wave of electronic commerce is going to require massive amounts of application integration across multiple organizations, as trading partners turn to the Internet to automate supply chains, forecasting systems, and government services, and new types of business interchanges.

This is potentially a big problem. If companies have difficulty integrating systems together within a single site, how are we going to do this on a broad scale over the Internet?

The way to overcome the first issue, and at the same time enabling the second, is to adopt an XML message-passing architecture to tie systems together. Common types of business data - whether it's a sales receipt, bank transfer, job application, or sales forecast - can be easily formatted in XML and sent between computers. BizTalk is going to do this by supporting standards, writing software and deploying successful solutions. Companies participating in BizTalk are committed to supporting XML in their products. The corporations participating in BizTalk are cranking up pilot projects that demonstrate this XML message-passing approach. Developers participating in BizTalk are publishing schemas to the website library and then using XML schemas to adapt their existing applications for electronic commerce.

Microsoft is strongly promoting the BizTalk concept. It provides a new e-business server, the BizTalk server, to allow the definition of trading agreements, edit schemata and define transformations between schemata. As such, the BizTalk server is a crucial component in the BizTalk philosophy.

As yet, support from other vendors is still limited. Also, there are a number of competing developments, amongst others UDDI (see section 9.5), that is also supported by Microsoft.

9.3 Electronic Business XML

The Electronic Business XML (ebXML) is an initiative, launched on December 1st 1999 [ebXML]. Its goal is the standardization of XML business specifications, to be achieved by the development of a technical framework, which enables XML to be consistently used for all business data. The project will run for 18 months after which it will be decided whether it is to become a long-term initiative. The mission of ebXML as specified on the official web page (see [ebXML]) of the project states:

> *“ebXML Mission: To provide an open XML-based infrastructure enabling the global use of electronic business information in an interoperable, secure and consistent manner by all parties”.*

The participation to the ebXML initiative is open and free to any organization and individual having technical and XML related expertise. The project was initiated by UN/CEFACT and OASIS (Organization for the Advancement of Structured Information Standards) and announced on September 15 1999.

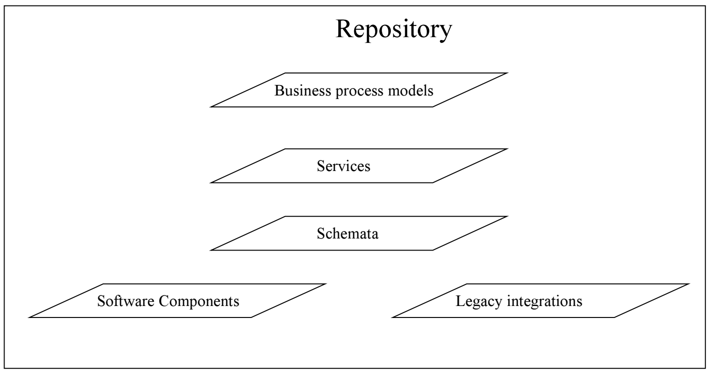
The goal behind ebXML is to build a framework, which can be used by different business applications to co-operate. The framework will represent a generic semantics of concepts and business processes from different market sectors, while allowing applications to communicate using XML or EDI syntax. The ebXML is not starting from scratch. It builds on the long-term experience with EDI syntax and semantics and on recent initiatives for translating EDI to XML. Moreover ebXML will organize existing XML schemas and DTDs into a general repository.

The current status of the project deliverables comprises several draft specifications open for comments from the general public. These are the Requirements specification, the Transport, Routing and Packaging Specification, and the initial specification of the architecture of the repository of business standards (see [ebXMLrs], [ebXMLrr], and [ebXMLtrp]). These documents present the very initial account of the various requirements and architecture of ebXML.

The ebXML framework is conceived as an XML-based platform which will support creation of enterprise applications capable of performing business services over the Internet. Such applications will be created from scratch using business process models (stored in a repository), integrated from software components (stored in a repository and conforming to model components from a repository), or will integrate existing applications to conform to XML-interfaces (again stored in a ebXML repository). The main ebXML parts are business process modeling methodology and metamodel, repository of reusable artifacts, and a messaging system. Each of them is described below.

1. *Business Process Modelling Methodology and Metamodel*. The metamodel will specify concepts for modeling business processes (in general and per sectors, e.g., roles, interactions, messages, data, services, transactions). The concepts and their visualization will be borrowed from existing works, e.g., UML, UN/CEFACT TMWG. After modeling of business models, services will be determined per business model. Services are defined in terms of possible interactions between business parties, which lead to performing a service. The interactions are specified in terms of document exchanges. Relations between interactions in a service are specified in terms of causality relation, alternatives, loops, parallels, etc. Any particular execution of a service is called a transaction.

2. *Repository of reusable artifacts.* The ebXML platform will define a repository of various reusable components. This repository will contain business process and service models, technical descriptions of these models (in terms of XML), software components conforming to the technical descriptions, and software components which integrate existing technical descriptions (e.g., EDI) with the XML technical descriptions and existing software systems to conform to the XML interfaces. The organization of the repository borrows ideas from the eCo framework. The difference is the stress on the business process and service modeling and the conformance between the layers in the repository. The layers in the repository can be depicted in the following way.



   - *Repository of business process and service models*: Using the methodology business process models will be created per sector and stored in a repository of business models. Various industry "verticals" will be required to specify their business processes and services using the methodology and concepts. Common business process and service components discovered through a comparative analysis of the models will be discovered and shall be submitted as candidates for "core business components". Such business components will help for the normalization of vertical business processes.

   - *Repository of technical descriptions*: The business model components contained in the repository will be mapped automatically (and using the metamodel) to XML-interfaces (in the form of DTDs and possible XML schemas). In the future a reverse-engineering techniques can be thought of to define the mapping (semi-automatically) from XML DTDs and Schemas to model components.

- *Repository of software components*: software components will be developed to conform to the technical descriptions in the repository. Since these descriptions will be expressed in XML the existing XML validation tools can be used to facilitate the automation of conformance testing between the software components and the corresponding XML interfaces. Software components will be certified as conforming to the XML interfaces (and the corresponding business processes and services) and stored in the repository.

- *Legacy integration*: Currently, there are no business models defined. Eventually existing methodologies and tools for business process modeling will be integrated into the platform. This will require translation between business models described using them and the selected modeling language and concepts. The repository of XML technical description can include not only descriptions derived from business models but also existing XML and EDI standards. In order to include also EDI standards translators from EDI to XML and vice versa are to be defined. Existing enterprise systems, such as accounting systems, ERP, etc. will be integrated to conform to the XML interfaces as well (instead of developing them from scratch). This will result in integration systems translating the output of the existing systems into XML documents and vice versa. Integration systems can be stored in the repository of software components as well.

3. *Messaging system.* The messaging system is an application, which manages the exchange of messages between parties. It does not depend on the content of the messages. In addition to the structure and semantics of business documents, the ebXML platform will define the envelopes and headers of business documents exchanged, and the protocols for exchanging of such documents. The focus will be on protocols for reliable message delivery and error handling, routing, secure delivery of messages, quality of service, etc. It is likely that the message envelopes and headers will be defined in XML as well. It is possible to use emerging XML standards for digital signatures and privacy preferences in the definition of the message envelopes (see [ebXMLtrp]).

The developments in ebXML seem to have wide support, and there are many relationships to current research at the Telematics Institute. ebXML, therefore, will be followed closely and actively over the years to come.

9.4 Simple Object Access Protocol (SOAP)

SOAP provides a simple and lightweight mechanism for exchanging structured and typed information between peers in a decentralized, distributed environment using XML. SOAP does not itself define any application semantics such as a programming model or implementation specific semantics; rather it defines a simple mechanism for expressing application semantics by providing a modular packaging model and encoding mechanisms for encoding data within modules. This allows SOAP to be used in a large variety of systems ranging from messaging systems to RPC.

SOAP consists of three parts:
1. The SOAP envelope construct defines an overall framework for expressing what is in a message; who should deal with it, and whether it is optional or mandatory.
2. The SOAP encoding rules defines a serialization mechanism that can be used to exchange instances of application-defined data types.
3. The SOAP RPC representation defines a convention that can be used to represent remote procedure calls and responses.

Although these parts are described together as part of SOAP, they are functionally orthogonal. In particular, the envelope and the encoding rules are defined in different namespaces in order to promote simplicity through modularity.

An example of a SOAP envelope is shown below. This example illustrates the use of a

```
POST /StockQuote HTTP/1.1
Host: www.stockquoteserver.com
Content-Type: text/xml; charset="utf-8"
Content-Length: nnnn
SOAPAction: "Some-URI"
<SOAP-ENV:Envelope xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
...
</SOAP-ENV:Envelope>
```

SOAP message within an http POST operation, which means the message is being posted or sent to someone. The example shows the namespaces for the envelope schema definition itself and for the schema definition of the encoding rules.

The draft version of the BizTalk Framework 2.0 of June 2000 has been defined to be SOAP 1.1 compliant, allowing BizTalk Framework XML documents to be transmitted in the form of SOAP messages [BizTalk].

9.5 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) is a new proposed standard that is developed by IBM, Microsoft, and Ariba [UDDI]. It is supported by several tens of leading IT companies, including CommerceOne, Dell, Sun and Tibco. UDDI uses standards-based technologies such as TCP/IP, HTTP, XML and SOAP to create a uniform service description format and service discovery protocol (Ariba, 2000). In Figure 9-1, the relationship between the different technologies is shown.

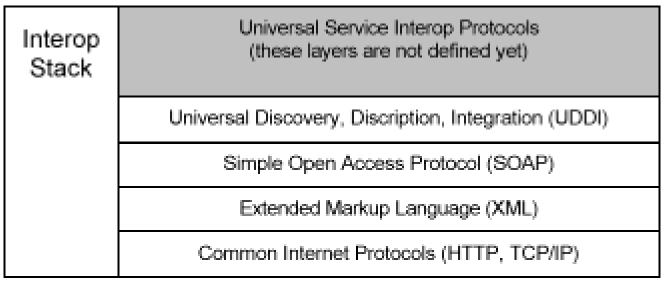

Interop Stack
Universal Service Interop Protocols
(these layers are not defined yet)
Universal Discovery, Discription, Integration (UDDI)
Simple Open Access Protocol (SOAP)
Extended Markup Language (XML)
Common Internet Protocols (HTTP, TCP/IP)

Figure 9-1: Layered view of WWW communication protocols

Information in the following sections is based on an overview of UDDI is presented in a technical whitepaper issued by the UDDI community [UDDIwp].

9.5.1 Overview

The Universal Description, Discovery and Integration specifications define a way to publish and discover information about Web services. The term “Web service” describes specific business functionality exposed by a company, usually through an Internet connection, for the purpose of providing a way for another company or software program to use the service. Web services are becoming the programmatic backbone for electronic commerce. For example, one company calls another’s service to send a purchase order directly via an Internet connection. Another example is a service that calculates the cost of shipping a package of a certain size or weight, so many miles via a specific carrier.

At first glance, it would seem simple to manage the process of Web service *discovery*. After all, if a known business partner has a known electronic commerce gateway, what’s left to discover? The tacit assumption, however, is that all of the information is already known. When you want to find out which business partners have which services, the ability to discover the answers can quickly become difficult. One option is to call each partner on the phone, and then try to find the right person to talk with. For a business that is exposing Web services, having to staff enough highly technical people to satisfy random discovery demand is difficult to justify.

Another way to solve this problem is through an approach that uses a Web services description file on each company’s Web site. After all, Web crawlers work by accessing a registered URL and are able to discover and index text found on nests of Web pages. The “robots.txt” approach, however, is dependent on the ability for a crawler to locate each Web site and the location of the service description file on that Web site. This distributed approach is potentially scalable but lacks a mechanism to insure consistency in service description formats and for the easy tracking of changes as they occur. UDDI takes an approach that relies upon a distributed registry of businesses and their service descriptions implemented in a common XML format.

9.5.2 UDDI business registrations and the UDDI business registry

The core component of the UDDI project is the UDDI business registration, an XML file used to describe a business entity and its Web services. Conceptually, the information provided in a UDDI business registration consists of three components: “white pages” including address, contact, and known identifiers; “yellow pages” including industrial categorizations based on standard taxonomies; and “green pages”, the technical information about services that are exposed by the business. Green pages include references to specifications for Web services, as well as support for pointers to various file and URL based discovery mechanisms if required.

9.5.3 Using UDDI

UDDI includes the shared operation of a business registry on the Web. For the most part, programs and programmers use the UDDI Business Registry to locate information about services and, in the case of programmers, to prepare systems that are compatible with advertised Web services or to describe their own Web services for others to call.

The UDDI Business Registry can be used at a business level to check whether a given partner has particular Web service interfaces, to find companies in a given industry with a given type of service, and to locate information about how a partner or intended partner has exposed a Web service in order to learn the technical details required to interact with that service.

10 XML: hype and hope

From this report, it should have become clear that XML is indeed a development that has tremendous momentum. Despite, or maybe: due to, its simplicity, the number of applications using XML is enormous. Not a single company in e-business dares not to talk about its relationship to XML.

Many of this is hype, yet the characteristics of XML do make it an important technology. Clear separation of structure, content and presentation/transformation makes it instrumental in defining and exchanging data standards, meta data, and data, and it can show of major importance in the field of enterprise application integration. The World Wide Web Consortium is standardizing the basic technology at a high pace. For now, the situation is that not all parts of XML have been thoroughly defined and that there exist some dialects of XML. This can be compared to the early days of HTML and will be behind us soon.

Yet, there is a downside to all this. Early semantic standardization trajectories, for example in EDI, took a lot of effort and time. They were initiated and carried out by standardization bodies that had a proven position in the field. Extensive negotiations were carried out to agree on underlying data models and to determine the messages. With XML, this has changed radically. Most of the important software vendors in e-business have defined their own standards, and are competing to become *the* standard. xCBL and cXML roughly do the same, but are supported by different groups. OFX is more of a documentation of what Checkfree and Intuit offer, than a solid standard. Also in the field of XML middleware many technologies compete. EbXML, UDDI, eCo: their similarities are much more prominent than their differences, which leads to a lot of confusion.

Given the fact that XML has this tremendous momentum, we expect this to be a temporary situation. Some standards will become dominant, probably those with the earliest, solid, tool support. Moreover, interoperability between these winners can be implemented using XML as well.

These developments together will help to bring the idea of worldwide market places, and flexible trading relationship closer to reality than other technologies, except for the Internet, have ever done. Therefore, XML gives hope indeed, although it generates confusion for the moment as well.

Every company seriously in e-business should invest in XML now and not wait. Despite all the different standards and disputes between them, they should obtain practical experience in XML and develop an idea on how XML can help in their communication with others, or in getting their internal data integration in place. The authors hope that this report has contributed in shedding some light on XML and its potential and limitations.

References

[Aho86]	Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman, *Compilers: principles, techniques and tools,* Addison-Wesley Publishing Company, 1986
[Apache]	http://www.apache.org
[BizTalk]	http://msdn.microsoft.com/xml/articles/biztalk/biztalkfwv2draft.asp
[Blom88]	Fred van Blommenstein, *Electronic Commerce, Internet en EDI*, Informatie, januari 1988, pages 14–25), (in Dutch)
[Bourret]	http://www.rpbourret.com/xml/XMLDatabaseProds.htm
[CheckFree]	http://www.checkfree.com
[Conn97]	Dan Connolly (red.), *XML — Principles, Tools, and Techniques*, The World Wide Web Journal, Vol. **2**, Issue 4, Fall 1997
[CSS]	http://www.w3.org/TR/REC-CSS1
[ebXML]	http://www.ebxml.org
[ebXMLrr]	*ebXML Registry and Repository Part1: Business Domain,* ebXML working draft May 11, 2000, version 1.0, http://www.ebXML.org/specindex.htm
[ebXMLrs]	*electronic business XML (ebXML) Requirements Specification,* ebXML Candidate Draft, April 28, 2000, http://www.ebXML.org/specindex.htm
[ebXMLtrp]	*ebXML Transport Routing and Packaging Overview and Requirements,* version 0-91, http://www.ebXML.org/specindex.htm
[FIX]	http://www.fixprotocol.org
[Hous95]	Walt Houser, *EDI Meets the Internet — Frequently Asked Questions about E-lectronic Data Interchange (EDI) on the Internet*, April 6, 1995, http://www.va.gov/ publ/standard/edifaq/index.htm
[Gold2000]	Charles F. Goldfarb, Paul Prescod, *The XML handbook,* Prentice Hall PTR, 2000
[HTML]	http://www.w3.org/MarkUp
[HTMDTD]	http://www.w3.org/TR/REC-html40/sgml/dtd.html
[IBMalpha]	IBM Alpha Works, http://www.alphaworks.ibm.com/
[IFX]	http://www.ifxforum.org
[IOTP]	Internet Open Trading Protocol, D. Burdett et al., http://www.ietf.cnri.reston.va.us/internet-drafts/draft-ietf-trade-iotp-v1.0-protocol-02.txt.
[OBI]	http://www.openbuy.org/
[OBI98]	OBI, *OBI White Paper*, available at http://www.supply.works.com/obi/white-paper.html
[OFX]	http://www.ofx.net/ofx
[OMG]	http://www.omg.org
[RDF]	http://www.w3.org/TR/PR-rdf-syntax/
[RFC-IOTP]	RFC for IOTP, http://www.ietf.org/internet-drafts/draft-ietf-trade-iotp-v1.0-protocol-06.txt
[Schema]	http://www.w3.org/1999/XMLSchema.dtd
[SemWeb]	http://www.w3.org/1999/04/WebData.html and http://www.w3.org/2000/01/sw/DevelopmentProposal
[SOAP]	http://www.w3.org/TR/SOAP
[UDDI]	http://www.UDDI.org
[UDDIwp]	http://www.uddi.org/pubs/Iru_UDDI_Technical_White_Paper.PDF
[UN/EDI]	http://www.unece.org/cefact
[XML]	http://www.w3.org/XML
[XMLEDI]	http://www.xmledi.com/