



MANATEE

IST-2001-38091

***Maritime Advanced Network for Anticipating Information
Technology Needs for e-work Environment in Safety at Sea***

Deliverable D2.1

Essential Technology for Markup Languages

Covering period 01.01.2003-30.11.2003

Report Version: 1.10

Report Preparation Date: 12-01-2004

Classification: Public

Contract Start Date: 01-12-2002

Duration: 24 Months

Project Co-ordinator: METTLE

Partners:	METTLE	France
	SP	Sweden
	BMT	UK
	AIM	Portugal
	MARINTEK	Norway
	EIS	Italy
	MCB	Italy
	HSB	Germany



**Project funded by the European Community
under the "Information Society Technology"
Programme (1998-2002)**

DELIVERABLE SUMMARY SHEET

Deliverable N°:	D2.1
Due date:	26-06-2003
Delivery Date:	26-06-2003
Classification:	Public

Short Description

This report gives an overall introduction and background to markup languages and XML. The application area is distributed systems and an important quality is security. The work is from a top-down perspective and focuses on functionality and possibilities but not on syntax and implementation issues.

Authors

Name	Company
Lars Strandén	SP
Tore Flobakk	Marintek

Internal Reviewing/Approval of report

Name	Company	Approval	Date
		Approved	

Document History

Revision	Date	Company	Initials	Revised pages	Short description of changes
1.1	12-01-2004	SP	LS	All 51	Minor word changes Structure discussion added
1.11	22-06-2005	HB	PAN		Classification changed

DISCLAIMER

Use of any knowledge, information or data contained in this document shall be at the user's sole risk. The members of the MANATEE Consortium accept no liability or responsibility, in negligence or otherwise, for any loss, damage or expense whatsoever incurred by any person as a result of the use, in any manner or form, of any knowledge, information or data contained in this document, or due to any inaccuracy, omission or error therein contained.

The European Commission shall not in any way be liable or responsible for the use of any such knowledge, information or data, or the consequences thereof.

Abstract

The report gives an overview of and introduction to the XML world i.e. the definition of XML and how it can be applied. The content shall not be considered a development tutorial since this is better covered by books and information available on the Internet. Instead this report takes a top-down perspective and focuses on the *possibilities* when using XML e.g. concerning functionality, tools and supporting standards.

Getting started with XML is very easy and the threshold is low. On the other hand since it is so low, many have actually started and created a vast number of standards (more than 450 so far) related to XML. Due to the high number another form of complexity emerge. One example is that when approaching the XML world it seems that everything depends on something else making it difficult to find a starting point. This is also true for the core functionality of XML.

Due to the complexity an important purpose of this report is to build up knowledge for the reader sequentially i.e. with a minimum number of forward references. In this way a more pedagogical approach is taken and enables further self studies. To guide these an extensive reference list is given at the end of the document.

This report shows the possibilities concerning information flow within distributed systems and where security is important. An important aspect is then the possibility of validation i.e. checking the written document according to a grammar. However, implementation aspects are not considered and so only the programming interface is discussed.

The result of this report is the description of what is possible to do with XML and what properties could be supported and expected. Also guidelines and recommendations are given and the most important recommendation is: *try to keep complexity down* otherwise maintenance and future extensions will be difficult to handle.

Executive summary

The deliverable D2.1 within the MANATEE project gives an overview of and introduction to markup languages and the XML world i.e. the definition of XML and how it can be applied. The report takes a top-down perspective and focuses on the *possibilities* when using XML e.g. concerning functionality, tools and supporting standards. The report focuses on *what* can be done but not on *how*.

XML stands for eXtensible Markup Language and is actually a *meta*-language because it just gives a framework for a specific XML based language (i.e. an XML application). Creating an XML application means that data has been *structured* in a particular way. This could be compared to HTML where the focus is on how to *present* data to the user. Both XML applications and HTML use *tags* for markup i.e. a start tag describing the data to come and an end tag signalling the end of data. An example shows the idea. Assume that we want to define a structure that makes it possible to relate vehicles with owners i.e. we want to create the corresponding XML application. Thus it is necessary to define person data, vehicle data and the relation between person and vehicle e.g. if a person owns more than one vehicle. We can then create an *instance* of the XML application, i.e. a specific record, for person A owning vehicle 1 and 2, another instance for person B owning vehicle 3 etc. This information could then be sent as files e.g. to authorities that will extract relevant pieces of information.

Getting started with XML is very easy since the knowledge threshold is low. On the other hand since it is so low, many have actually started and created a vast number of standards (more than 450 so far) related to XML. Due to the high number another form of complexity emerges. One example is that when approaching the XML world it seems that everything depends on something else making it difficult to find a starting point. This is also true for the core functionality of XML.

One strength of XML is that it allows the creation of an application specific *grammar*, i.e. a list of rules how to structure data, which can be used by a parser to validate a specific instance of an XML application. Two major types of grammars exist, the DTD and the XML Schema, where the latter is probably the winner in the long run since it is much more capable. By using a grammar we could check if data has been recorded correctly otherwise an error is generated.

Apart from aspects mentioned above there are several standards that address core XML functionality and the use of some of them is listed below:

- to transfer data using XML based protocol at the highest level
- to extract or merge pieces of information
- to include encryption in order to hide sensitive pieces of data
- to include authentication in order to guarantee that information is generated and handled by authorized persons
- to make transformations to/from other XML applications
- to make presentation of data

The scope of the deliverable D2.1 includes distributed systems and handling of security issues. For distributed systems we have to use standardised communication protocols at lower levels, such as TCP/IP, in order to transfer data correctly. For security we have the following general aspects:

- *integrity* – how do protect data from being manipulated

- *confidentiality* – how do we keep information secret
- *authentication* – how do we know for sure who has sent the information
- *authorisation* – how much is an authenticated person allowed to do
- *non-repudiation* – how can we verify that information actually has been sent, how can we verify that information actually has not been sent

The definition of an XML application is to a large extent dependent on the underlying data model; what kind of data is relevant and how are relations between different pieces of data defined. Thus a thorough analysis is necessary before the XML application is specified.

The result of this report is the description of what is possible to do with XML and what properties could be supported and expected. Also guidelines and recommendations are given and the most important recommendations are: *try to keep complexity down* and *prepare for extensions* otherwise maintenance will be difficult to handle.

Contents

1	Introduction	10
1.1	Background	10
1.2	The markup idea	10
1.3	XML background	11
1.4	The XML standard	13
1.5	XML applications	13
1.6	Organisations	15
1.7	Disclaimer	18
2	Definitions and terminology	19
3	Scope	21
3.1	General	21
3.2	Covered topics	21
3.3	Not covered topics	22
4	XML	23
4.1	Introduction	23
4.2	Valid vs. well-formed	23
4.3	XML processor	24
4.4	Logical structure	24
4.5	Physical structure	28
4.6	Use of URI	29
4.7	Naming conventions	30
4.8	Namespace	30
4.9	Reserved attributes	32
4.10	XML Infoset	32
4.11	Metadata	33
4.12	Conclusions	33
5	XML support	34
5.1	Introduction	34
5.2	Processor	34
5.3	Relative resource addressing	34
5.4	Addressing fragments of resources	35
5.5	Parsing fragments of resources	38
5.6	Transform XML application instances	39
5.7	Structuring resources	40
5.8	Merging XML application instances	43
5.9	Presentation	45
5.10	Comparing XML application instances	46
5.11	Script languages	47
5.12	Conclusions	47
6	Grammar	48
6.1	Introduction	48
6.2	DTD	48
6.3	XML Schema	49
6.4	Alternatives	51
6.5	Conclusions	52
7	System aspects	53
7.1	Introduction	53

7.2	Transaction	53
7.3	Data transfer	54
7.4	Databases	56
7.5	Web services	57
7.6	Security	58
7.7	User interface	62
7.8	Application Programming Interface	62
7.9	Conclusions	64
8	XML tools	65
8.1	Introduction	65
8.2	Development of XML Application Instance	65
8.3	Development of grammar	66
8.4	Development of stylesheet	66
8.5	Presentation	66
8.6	Data access (API)	66
8.7	Tool links	66
8.8	Conclusion	67
9	Creating an XML application	69
9.1	Introduction	69
9.2	Data model	69
9.3	Overall design principles	69
9.4	Use of design patterns	70
9.5	Preparing for extensions	73
9.6	Documenting an XML application	75
9.7	Checklist before making things too complex	76
9.8	Making the XML application a standard	76
9.9	Conclusions	77
10	Conclusions	78
11	References	80

Preface

The content of this report concerns general functional and security aspects in a possibly distributed environment i.e. information shall be transferred in a secure way between clients and servers. The intentions of the work are:

- to be pedagogical e.g. by including figures and especially by minimizing the number of forward references
- to be clear and concise e.g. concerning definitions and structure
- to take a top-down approach starting at defining the scope and then going into details
- to give hints for further studies
- to describe the supporting infrastructure around XML
- to transfer experiences and lessons learned
- to describe resources for more information
- to be reasonably complete concerning relevant overall aspects

The report could be used as a pragmatic primer for understanding the XML context and its world of standards, abbreviations, recommendations, guidelines etc. A typical reader could be a person considering creation of a new XML application or anyone interested in understanding the world of XML. However, actual detailed instructions are not included. These could instead be found using the references given in this report. Also a beginner could appreciate this report, however, it is probably better to first read introductory information concerning how to actually write an XML document and to see some examples. The purpose of this report is to describe *what* is possible but not focus on *how*. For the latter there are many tutorials and books available.

The layout is as follows.

In chapter 1 a general introduction is made of markup languages and an XML background is presented.

Definitions and terminology are described in chapter 2. Comprehensive glossaries can be found in the literature and on the Internet. Here are only those terms included that are necessary for this report.

Chapter 3 contains the scope and purpose of the report; what is included and what is not.

Since the purpose is not to create a development tutorial but instead focus on the possibilities, a functional discussion is made in chapter 4 concerning the capabilities and structures of XML.

Chapter 5 contains discussions of extended functional support defined by generally accepted standards. The description could be seen as a principle toolbox that sets the scope of allowed possibilities.

In chapter 6 the focus is on how to use grammars in order to verify that documents follow specific rules. Two main competitors are presented: DTD and XML Schema.

Chapter 7 describes the context of XML when used in applications, in principle how information is transferred between XML documents and the application.

For use of XML efficiently, commercial tools are necessary. A principal overview is given in chapter 8 describing capabilities and focusing on tool types.

Based on discussions given in the previous chapters, general guidelines are described in chapter 9 how to create an XML application in a top-down manner.

Chapter 10 contains conclusions regarding the top-down handling of XML applications.

Finally chapter 11 gives references used for this report and for further reading.

1 Introduction

1.1 Background

This report is one of the results of work package 2 within the EU FP5 MANATEE project (IST-2001-38091). The reason for this report is to create a common understanding of markup languages in general and XML in particular. This report will then be the basis when creating a markup language for maritime safety based on XML; the Maritime Safety Markup Language (MSML). The results of this report will also be used for guidance e.g. concerning system design and principal implementation issues.

1.2 The markup idea

The markup idea is simple:

take a text document, containing some kind of data, and provide extra text information (markup tags) in order to make it possible to interpret the data.

Of course it is important to keep the data and the markup information clearly separate. It is practical to let the markup information be ordinary text and thus we end up with a document where both data and markup information are readable as ordinary text. Thus the markup world actively supports the principle “write once read many”. Normally, markup is made at the beginning and at the end, using specific tags, of the information to be marked:

```
<start markup>data to be marked<end markup>
```

Which kind of documents could be marked up? In principle any document containing information that we want to *describe* and where the description in itself is of value for another person, computer, system etc. For example, we could mark up a novel with markup tags describing:

- Dialogs
- Thoughts
- Descriptions of environment
- Information necessary for a play or movie
- etc

As another example, we could mark up source code with markup tags describing:

- Type declarations
- Variables
- Function declarations
- Function calls
- etc

When marking up source code we could e.g. have a function declaration as

```
<function name="xxx" returnType="integer"> ... </function>
```

where the markup is named “function” and ends with </function> (as in XML). In the XML world the “function” markup is called an *element* and “returntype” above is an example of an

attribute. This element declares a function with a specific name and type of return value as attribute. A call of such function could be marked up as

```
<call function="xxx"> ... </call>
```

The information between the start tag and end tag is called the *element content*. For this example it could contain zero or more parameters with names and types such as:

```
<parameter name="yyy" type="zzz"> ... </parameter>
```

This allows for nested/structured parameters (like structs in c).

From these examples we see that there are no limits imposed from the beginning. Instead we just have to focus on the actual use of the marked up document. We also see that it is possible to create a tree representation using the elements as nodes in the tree.

The next thing to consider is in what way the markup information shall support the interpretation and what the interpretation shall be used for. We give two examples here:

- to define the presentation of data (like HTML)
- to define the type of data e.g. how data shall be handled by an application (like XML)

One example of the first case could be:

```
<Headline>Introduction to Markup Languages</Headline >
```

and of the second case:

```
<Engine Speed>6220</Engine Speed>
```

Note that for the second case we still have to think of how data shall be presented.

So far so good. If we now have a marked up text document what do we want to do with it? We can list a number of uses that would be desirable:

- to transfer structured data to users
- to interpret data in the intended way
- to modify and add information using manual or computer generated actions
- to search and extract pieces of information
- to include encryption in order to hide sensitive pieces of data
- to include authentication in order to guarantee that information is generated and handled by authorized persons
- to have a tool that could check that the document fulfils specific design rules
- to transmit the document in a secure way i.e. without any unauthorised persons affecting the content or being able to read it
- to have a nice looking presentation of data and to have it look the same independent of the computer presenting it
- to follow standards as far as possible in order to have a more stable handling and to have an easier exchange with others

All these, and more, will be discussed below in more detail.

1.3 XML background

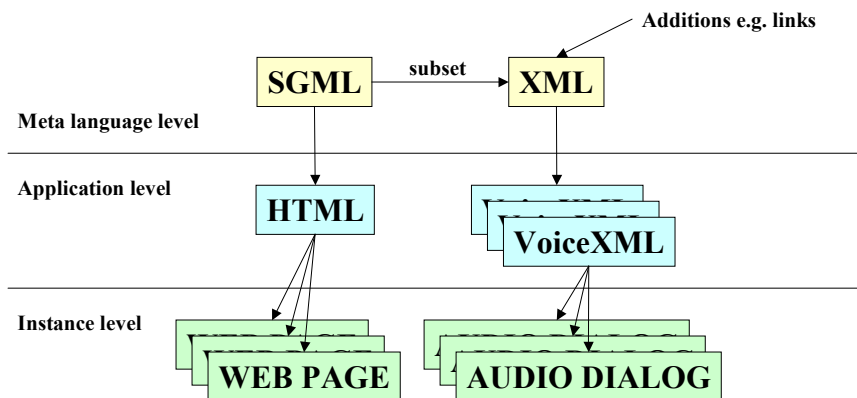
The historical starting point is SGML (Standard Generalized Markup Language), which was officially defined 1986 as the ISO 8879 standard. SGML was indeed general, actually too

general for many uses. Thus a need for something simpler emerged and for classifying data the answer was XML (Extensible Markup Language) with the first version from 1998. Both SGML and XML are widely accepted and there are no real alternatives to them. Note that the current XML version is still 1.0 however categorised as the second edition. The purpose was to have “80 % of the SGML possibilities with a size of 20 % of the SGML specification”.

For understanding what SGML and XML stand for the following is crucial:

SGML and XML are generalized (or meta) languages i.e. they have to be instantiated in order to get a useful markup language.

SGML and XML should be seen as empty shells that have to be filled before practical use is possible. Thus, if someone says that he/she is using XML it does not help you much since you must know *how* XML has been used in order to apply it. We will from now on use “*XML application*” to denote a specific language created using the general rules given by “*XML*” (i.e. the meta language). Once the XML application is defined it can be used as a concrete framework for documents containing specific data. In this report each document containing a specific set of data fulfilling a specific XML application will be called “*XML application instance*” or just *instance* if no misinterpretation is possible. The reason why we use instance here is that different and unclear notations are sometimes used in the literature and *instance* really describes what it is. The picture below shows this where VoiceXML is one of many existing XML applications today.



The most important difference between HTML and XML applications (i.e. at the language level) is that HTML is a fixed standard (for presenting information) while XML applications can be created by anyone interested in a specific application area.

The focus of this report is on the meta language XML and how it makes it possible to define different XML applications where for each, many different XML application instances can be created. Put in another way, from the creation perspective we have:

- XML is created by the organisation W3C and is a fixed world standard.
- XML application is created by anyone interested in a specific application area. The XML application is defined by rules given by a separate grammar such as DTD, XML Schema (the discussion of these will be postponed until later).
- XML application instance defines a specific item within the application area.

We give an example now. Assume that it is necessary to create an XML application that makes it possible to describe vehicles in a standardised way. The reason could e.g. be that authorities need such information for statistic analysis. The XML application should thus make it possible to handle any type of vehicle in the same manner. What should the XML application then make it possible to specify? There are many possibilities and we give some examples below:

- The type of vehicle e.g. car, bus, truck.
- That there is *a single* owner of the vehicle.
- That the owner could own more than one vehicle.
- That there have been zero or more owners before the current one.
- The set of allowed characters for the licence plate.

We can then create individual instances for car 1, truck 14 etc each fulfilling the rules of the XML application. This example shows the typical use of XML applications. By including rules, what is allowed and not, makes it possible to have more focused documents and to design them in a standardised way. Of course it will then be important to try to make the XML application officially accepted as soon as possible so no other will invent another XML application (probably slightly different) with the same purpose! This could be a real threat.

How instances are actually handled by computers and humans will be discussed later on.

1.4 The XML standard

The XML standard is a W3C recommendation (see [6]) but denoted a standard in this report (because it is).

The standard is not the first document to read. It is a reference document in the meaning that information at the beginning refers to information later on i.e. one has to know it before it can be studied! There are many, much more pedagogical documents for learning XML. However, by using the online version of the XML standard together with a paper version the reading is simplified. One could then use definition links in an easy way.

An alternative is to use the annotated standard see [27].

The most important aspect to remember when studying the standard is that it is focused on requirements on a tool for handling XML application instances. The notation is specified in Backus-Nauer form and chapter 6 of the standard defines it. If this form is not known to the reader chapter 6 is the first chapter to study.

Also note that there are (informal) guidelines included in the standard since the Backus-Nauer specifications in some cases allow too much freedom.

1.5 XML applications

1.5.1 Introduction

In this chapter we give some examples of XML applications (i.e. languages) already defined for *specific application areas*. From e.g. [2] we could see that a lot of XML applications exist.

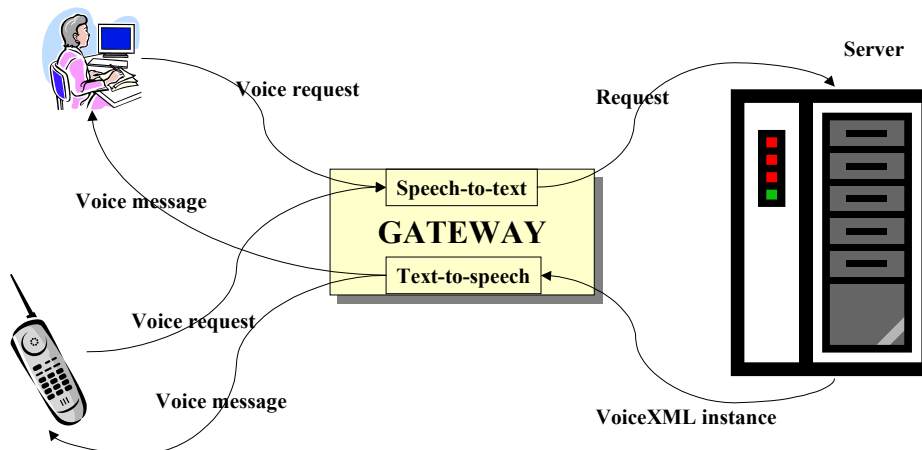
There are also other XML applications used for *supporting* the XML infrastructure but these are not considered here (this will be done later).

The list of XML applications presented below is by no means complete nor should it be considered representative for an overall survey. The purpose is just to show the diversity of XML applications.

1.5.2 VoiceXML

For the current specification see [8]. It contains ca 170 pages. VoiceXML is supported by W3C.

VoiceXML is used for making it possible to transfer audio information as text on the Internet between different implementations such as PCs, mobile phones etc as shown in the figure below. A requirement of VoiceXML is that the HTTP protocol must be supported.



VoiceXML resembles a programming language. Among other things it contains constructs like:

- GOTO
- IF-ELSE-ELSEIF
- Subroutine
- Variable declaration
- Loop
- Means for specifying platform specific implementation data

This imposes data and control flow and thus execution/simulation of dialogs is possible.

1.5.3 SMIL

For the current specification of Synchronized Multimedia Integrated Language (SMIL) see [9]. It contains ca 480 pages. SMIL is supported by W3C.

The purpose is to enable interactive multimedia presentations with a special focus on timing issues. SMIL is used for classifying information. SMIL includes a number of modules:

- Animation Modules

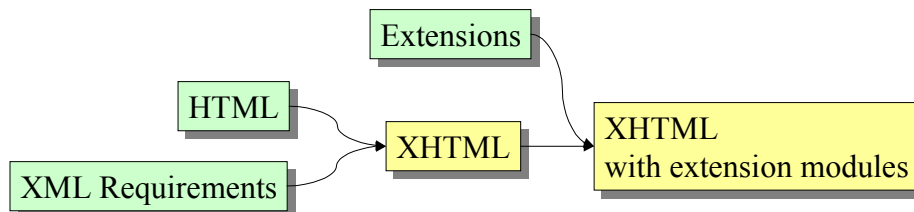
- Content Control Modules
- Layout Modules
- Linking Modules
- Media Object Modules
- Metainformation Module
- Structure Module
- Timing and Synchronization Modules
- Time Manipulations Module
- Transition effects Modules

Browsers e.g. Internet Explorer support SMIL.

1.5.4 XHTML

For the current specification of Extensible HyperText Markup Language (XHTML) see [10]. It contains ca 20 pages. XHTML is supported by W3C.

The purpose is to fit HTML into XML by including extra requirements. These are not many and not severe and the major benefit is that presentation and classification of data can be handled the same way e.g. making it possible to use the same XML tools. Important is also the possibility of extensions in XHTML which are not possible to do in HTML. XHTML is based on HTML 4.



The drawback of XHTML is of course that HTML is since long a worldwide established and used standard and changing to XHTML cannot always be motivated. XHTML has the same purpose as HTML but being more extensible since XHTML is an XML application.

1.5.5 SVG

For the current specification of Scalable vector graphics see [30]. It contains ca 720 pages. SVG is supported by W3C.

The purpose is to make it possible to transfer graphics represented by vectors instead of e.g. using bitmaps which require much more data. Of course pictures will not be suitable for SVG but e.g. drawings will be much more effectively transferred using SVG.

1.6 Organisations

1.6.1 Introduction

The list below gives some examples of the most important XML related organisations.

1.6.2 World Wide Web Consortium (W3C)

The World Wide Web Consortium (see [40]) is the natural starting point for all aspects concerning XML. W3C started in 1994 and has around 450 members (see <http://www.w3.org/Consortium/Member/List>).

Four activity domains are defined:

- Architecture
- Interaction
- Technology and Society
- Web Accessibility Initiative

From <http://www.w3.org/Consortium/#mission> the long term goals of W3C are defined as:

1. *“Universal Access: To make the Web accessible to all by promoting technologies that take into account the vast differences in culture, languages, education, ability, material resources, access devices, and physical limitations of users on all continents;*
2. *Semantic Web: To develop a software environment that permits each user to make the best use of the resources available on the Web;*
3. *Web of Trust: To guide the Web's development with careful consideration for the novel legal, commercial, and social issues raised by this technology. “*

Official W3C specifications are classified as recommendations but should be considered as standards. A comprehensive list can be found at <http://www.w3.org/TR/>.

Tutorials can be found at <http://www.w3.org/2002/03/tutorials>. There is also an online school at <http://www.w3schools.com/>.

1.6.3 Dublin Core Metadata Initiative

From <http://dublincore.org/> we have:

“The Dublin Core Metadata Initiative is an open forum engaged in the development of interoperable online metadata standards that support a broad range of purposes and business models. DCMI's activities include consensus-driven working groups, global workshops, conferences, standards liaison, and educational efforts to promote widespread acceptance of metadata standards and practices.”

There are 15 document elements defined (see <http://dublincore.org/documents/dcmi-terms/>):

- Title
- Creator
- Subject and Keywords
- Description
- Publisher
- Contributor
- Date
- Resource Type
- Format
- Resource Identifier

- Source
- Language
- Relation
- Coverage
- Rights Management

Using these metadata (i.e. data about data) makes it possible to express, in a standardised way, information of documents on the web.

1.6.4 OASIS

From <http://www.oasis-open.org/> we have:

“OASIS is a not-for-profit, global consortium that drives the development, convergence and adoption of e-business standards”

and from <http://xml.coverpages.org/> we have:

“OASIS provides the Cover Pages as a public resource to document and encourage the use of open standards that enhance the intelligibility, quality, and longevity of digital information.”

In June 1999 OASIS created XMLORG (see <http://www.xml.org/>):

“Our mission is to accelerate the global utilization and adoption of XML by providing an open and Non-profit industry portal that brings together all members of the XML community, including technologists, developers, and businesspeople.”

“The Cover Pages is a comprehensive Web-accessible reference collection supporting the SGML/XML family of (meta) markup language standards and their application. The principal objective in this public access knowledgebase is to promote and enable the use of open, interoperable standards-based solutions which protect digital information and enhance the integrity of communication. A secondary objective in The Cover Pages is to provide reference material on enabling technologies compatible with descriptive markup language standards and applications: object modeling, semantic nets, ontologies, authority lists, document production systems, and conceptual modelling.”

OASIS is the natural starting point for trying to establish a new XML application as a standard.

1.6.5 Apache

From <http://xml.apache.org/> we have the goals for the Apache XML Project as:

- *“to provide commercial-quality standards-based XML solutions that are developed in an open and cooperative fashion,*
- *to provide feedback to standards bodies (such as IETF and W3C) from an implementation perspective, and*
- *to be a focus for XML-related activities within Apache projects”*

Within the Apache project a number of XML tools have been developed.

1.6.6 SGML/XML Special Interest Group

This fundamental group is within W3C and can be found at <http://www.w3.org/XML/SIG/> and from there we get:

“The XML Special Interest Group is a forum for the discussion of issues relating to the exchange of structured information over the World Wide Web. In particular, it serves as a resource for the communication of expert opinions to the XML Working Group. All activity of the XML SIG takes place on an archived mailing list.”

1.7 Disclaimer

Some of the W3C recommendations given in the reference chapter are draft versions and are likely to be modified. Thus, before actual development work starts the used recommendations must be checked. Also some of the W3C recommendations are new and thus e.g. tool support may be poor.

Further, some web references might become obsolete. Try searching the title on Altavista, Google for finding a valid link.

2 Definitions and terminology

There are many XML glossaries available on the Internet and in books. These can be searched for using search engines such as Altavista, Google. Below are those listed that are specific to this document and important for the understanding of this report.

API	Application Programming Interface
Application	Applications means an application within an area of interest e.g. maritime applications. Do not confuse with the use of application in “XML application”, see below.
COTS	Commercial Off The Shelf i.e. a standard product that could be bought.
DTD	Document Type Definition (inheritance from SGML), a specification means for the allowed structure of the XML application instances i.e. a grammar. Note that DTD does not conform to XML.
Grammar	Here, grammar denotes the rules, defining the XML application that can be used for validating the corresponding instance. The grammar is defined using DTD, XML Schema or similar.
GUI	Graphical User Interface
HTML	HyperText Markup Language used for presenting information
IDE	Integrated Development Environment
Instance	or XML application instance is a document containing specific data and fulfilling the rules given by the corresponding XML application.
Infoset	An abstract data set defining the possible constituents of an XML application instance e.g. attribute, element, comment
IRI	Internationalised Resource Identifiers is an extension of URI.
Media type	According to RFC 2376 (for XML) (http://www.ietf.org/rfc/rfc2376.txt)
Metadata	Data about data. One example is the Dublin Core Metadata Initiative (http://dublincore.org/documents/dcmi-terms/)
Meta language	A language giving the rules for further specifications. Here the main interest is in the meta language XML making it possible to define different XML applications
Normative	Required, when discussing standards and W3C recommendations.

OO	Object Oriented
Parse	To let a tool analyse text and identify different parts of it for further handling (here according to the XML infoset). For example, identify attributes, elements etc.
Root	The root, of an XML application instance, is not visible but corresponds to the document entity and contains a single top element (if well formed)
Resource	Something accessible via a URI
RPC	Remote Procedure Call
SOAP	Simple Object Access Protocol
URI	Universal Resource Identifier
URL	Universal Resource Locator
Valid instance	An XML application instance is valid if it fulfils the grammar of the XML application.
Well-formed instance	An XML application instance is well-formed if it fulfils the rules given in the XML specification.
XML	eXtensible Markup Language
XML application	DTD, XML Schema or other grammar specifying elements etc for a specific type of instantiation of XML. This corresponds to vocabulary.
XML application instance	See Instance
XML Infoset	See Infoset
XML Schema	A specification means for the allowed structure of the XML application instances i.e. a grammar. XML Schema conforms to XML.
W3C	World Wide Web Consortium

3 Scope

3.1 General

There are so many standards around and it is not realistic to cover them all and not even to categorise them in a consistent manner. Instead the focus of this report is the pragmatic *use* of XML. Thus the scope includes aspects that are related to “best practise” instead of trying to be complete. One important goal of this report is to list and describe the most useful pieces of functionality in the XML world. Expressed in another way:

This report focus on what is possible but not on how to implement it.

Generally we include in the scope:

- how to develop XML applications and their instances
- how to treat the created XML application instances
- how to transfer XML application instances in computer networks (especially via the Internet)
- how to handle properties

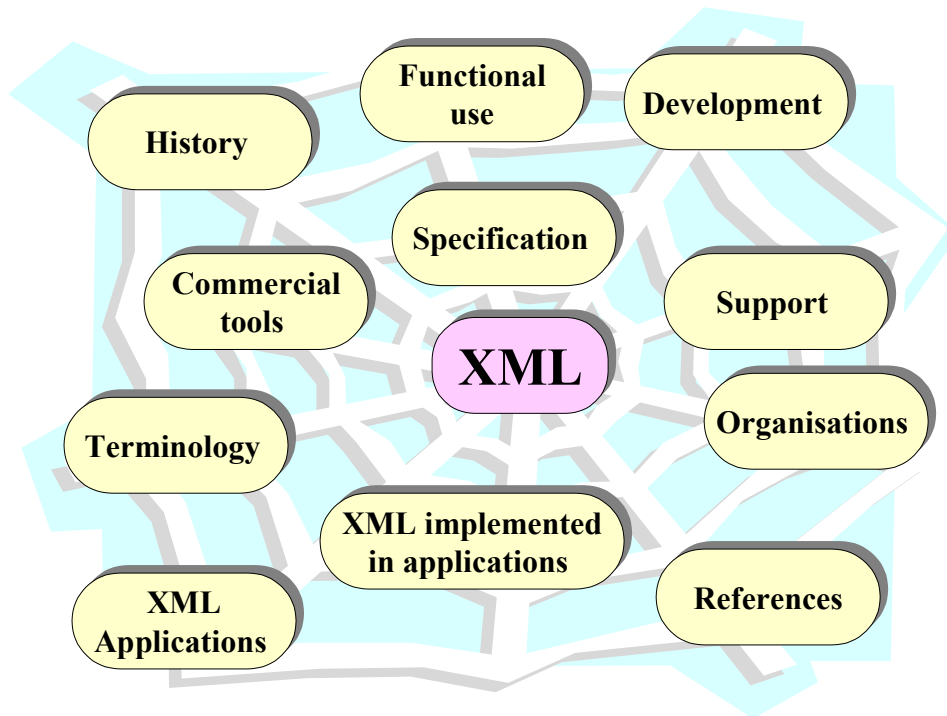
The result after studying this report should help the development of XML applications that will be of practical use and importance and at the same time avoiding most (hopefully all) of the pitfalls. The development details can be found via resources and references given in this report.

The type of application areas addressed here are characterised by:

- Distributed system i.e. there is a need for sending and receiving data documents.
- Security aspects i.e. integrity, authorisation, authentication, confidentiality, non-repudiation (to be able to verify that an action has taken place irrespectively of what persons claim).
- Possible safety aspects i.e. no harm to persons, equipment and environment.
- Possible real-time aspects e.g. information delivery before a specific deadline

3.2 Covered topics

The starting point is XML as such. In the picture below the web connects the related XML aspects included in this report.



The picture is motivated by:

- *History* – the XML background (covered earlier in the introduction)
- *Specification* – how XML is specified
- *Development* – how development of an XML application should be made.
- *Functional use* – how the XML application shall be used, the functions to include and the functional support
- *Commercial tools* – what can they do, how do they support?
- *Support* – what are the XML resources?
- *Organisations* – which exist, what do they do (this also includes standardisation committees)
- *Terminology* – important in order to speak the same language
- *Existing XML applications* – different types, what are they good at?
- *XML implemented in application areas* – how are XML application instances principally handled in applications
- *References* – the information used for this report and for further studies

3.3 Not covered topics

The following aspects are not included in this work:

- Organisation issues
- Work procedures
- Applications not using validation
- Safety (since it depends completely on the application and its use)
- Fault handling (since it depends completely on the application and its use)
- Implementation (only the interface to the implementation is considered)
- Syntax (only used in examples and for specific explanations)
- Protocols and protocol bindings

4 XML

4.1 Introduction

The first point to note is that the current XML standard version (see [6]) is still 1.0 (although second edition is in use today) in spite of the extensive use of the standard since 1998. This could be a quality mark concerning the scope, the stability and the quality of the document. However, it could also indicate that the recommendation is too general and/or the scope is too limited. In any case, for a user or XML application designer it means that there is a stable ground to stand on.

On the other hand, how come there are so many standards and recommendations in the XML world? It is really difficult to get a clear picture of what is necessary, recommended or irrelevant. Hopefully this report will make the survey somewhat easier. A help is to follow the historical evolution. In this way it is e.g. easier to understand the relation between DTD and XML Schema and the way of using them in an XML application instance.

In this chapter it is important to remember that DTD is actually a part of the XML standard. This explains why DTD is mentioned at several places below but not other grammars such as XML Schema. In the XML standard, DTD is not considered a document on its own. Instead it is more described as a necessary support for validation.

Some general reasons for all the different XML applications are:

- The threshold for getting started with XML is low. Many good sources of information exist.
- When many XML applications are defined there is a need for structuring them at a higher level, thus introducing more XML based documents.
- Initiatives come in many cases from vendors outside the control of W3C.
- XML application instances contain plain text which makes it possible to discuss them without specific tools and without highly experienced persons involved “everybody could have an opinion”.

As a general comment, it is important that rules and definitions from W3C should be followed in order to not complicate matters e.g. when commercial tools are used.

Note that the purpose here is not to give a tutorial how to make XML applications and their instances. Good tutorials exist as books (see e.g. [1]) and as downloadable information from the Internet. Instead the purpose here is to present the constituents, how they interact and guidelines how to handle them.

4.2 Valid vs. well-formed

As mentioned earlier we will only consider XML application instances that could be validated. Validation is made by checking the instance against rules defined using DTD, XML Schema or other type of grammar representation. If not using validation only well-formedness could be checked i.e. using general rules given by XML as such and this is normally on a too low level.

In order to not complicate matters we will postpone the discussion of validation until later but at this point only stress that it is possible to have rules for how to write XML application instances and fulfilling these rules could be checked.

4.3 XML processor

The notion of XML processor is introduced in the XML standard. The XML processor describes a piece of software used for processing XML application instances. The main work of the XML processor is to parse the XML application instance and present any errors. There are two basic types of XML processors:

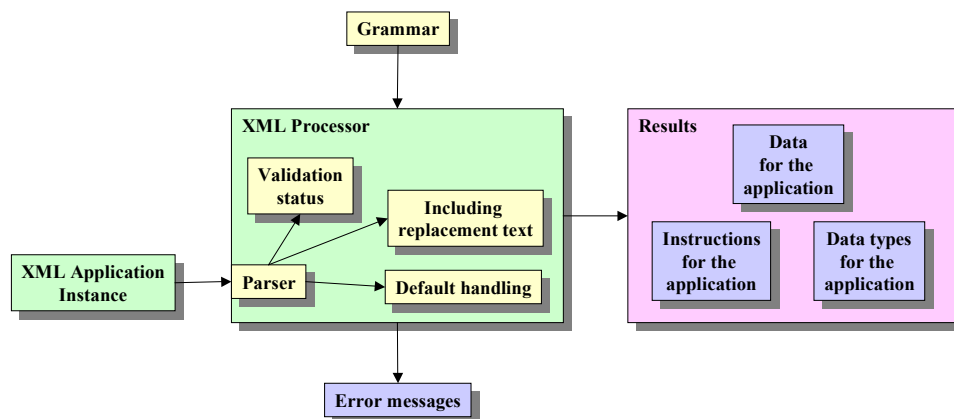
- Non-validating – can only check well-formedness.
- Validating – can check well-formedness and validity (using an XML application specific grammar)

The XML processor could parse using one of the following principles:

- Document-based parsing i.e. the whole XML application instance is stored before parsing takes place.
- Event-based parsing i.e. parsing is made successively as the XML application instance is read.

Other types of XML processors are also possible e.g. those unaware of namespaces.

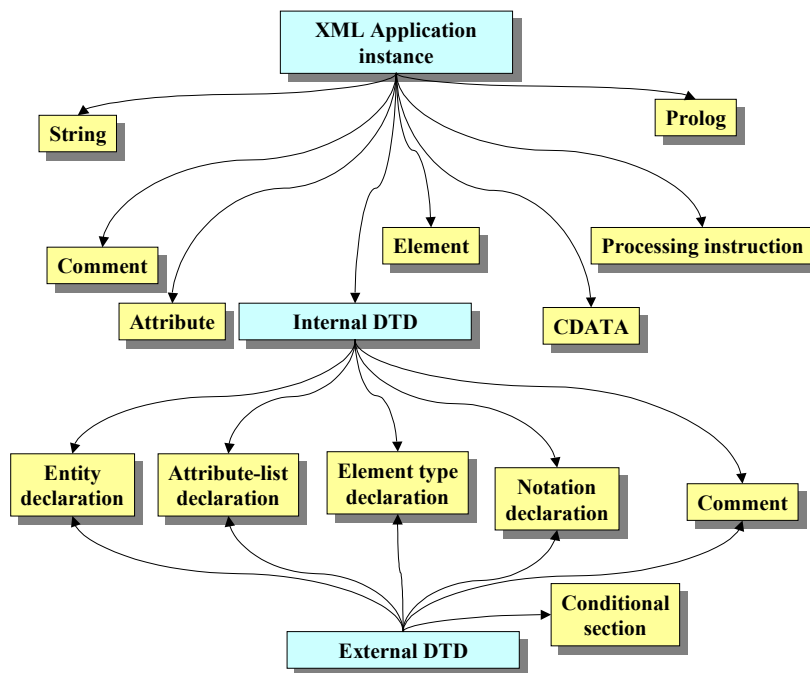
There are many rules described in the XML standard e.g. in what order replacement texts should be handled (a “macro” can be defined that gives a name for a larger replacement text), what kind of information that shall be passed to the application unchanged, default values etc. The application is the consumer of information given by the XML processor and represents a specific area of interest. The picture below shows the principal inputs and outputs and shows the relation to the application.



The XML application instance could be created manually or by a computer and the results are used by a computer within a specific application area. Note that the picture shows logical constituents i.e. nothing is said about where the items are placed physically e.g. they could all be placed on a single computer or distributed across the Internet.

4.4 Logical structure

The logical structure describes the constituents of an XML application instance and how they are related.



The picture is described successively in the text below.

An XML application instance consists of the following parts:

- A prolog containing the XML declaration and the document type declaration
- A single document element (containing all child elements)

An element is the basic markup constituent and the document element can be seen as the root of the specific area of interest. The XML declaration gives some overall administrative information such as the used XML version. One example is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
```

where the XML version is specified together with the encoding of characters and if the document type declaration is included in the XML application instance or not (see below). The document type declaration points to a DTD that contains the rules (the grammar) that are specific for the XML application. The DTD could be external or internal i.e. contained in the XML application instance within the document type declaration. However, this is less flexible since e.g. if the grammar shall be changed all XML application instances must be modified. Also, extra space is needed in every instance. The choice of internal/external is defined in the prolog (standalone above). The document type declaration for the not contained version could look like:

```
<!DOCTYPE vehicle SYSTEM "vehicle.dtd">
```

where `vehicle` is the name of the top element and `vehicle.dtd` is the name of the corresponding DTD. The top element and its child elements could look like (a comment is marked with `<!--` and `-->`):

```

<vehicle>                                <!-- Top element -->
  <type>                                  <!-- First child element -->
    car
  </type>                                 <!-- End of first child element-->
  <colour>                                <!-- Second child element -->
    white
  </colour>                               <!-- End of second child element-->
</vehicle>                               <!-- End of top element -->

```

An element markup must have an ending markup (the same name preceded by `/`). Putting this together we get the XML application instance for a white car (informally belonging to the application area `vehicle`) as:

```

<xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE vehicle SYSTEM "vehicle.dtd">
<vehicle>                                <!-- Top element -->
  <type>                                  <!-- First child element -->
    car
  </type>                                 <!-- End of first child element-->
  <colour>                                <!-- Second child element -->
    white
  </colour>                               <!-- End of second child element-->
</vehicle>                               <!-- End of top element -->

```

The next thing to consider is *attributes* i.e. parameters that could be specified in connection with elements as name-value pairs. We have already seen examples above:

- `version="1.0"`
- `encoding="UTF-8"`
- `standalone="no"`

In the example above we could replace the `colour` element with the corresponding attribute.

```

<vehicle>
  <type colour="white">
    car
  </type>
</vehicle>

```

Note that attributes do not affect (tree) structure but elements do. In principle we also have a third possibility; to include the attribute in the element name i.e.

```

<vehicle>
  <white_car>
  </white_car>
</vehicle>

```

In this simplified case we get an empty element. Even if more compact, the flexibility is gone and thus this solution is generally of less interest than the other two.

A discussion can be found in the literature if child elements or attributes shall be used. Child elements improve the tree structure but could give unnecessary overhead. A strict rule is not possible to give and the choice must instead be based on experience and on expected future use e.g. concerning extensions. However, some general guidelines are given below:

- Child elements are directed more towards the human reader. Attributes are directed more towards machine interpreters.
- Child elements extend the tree structure while attributes do not. For future extensions it is easier to expand a tree structure than to handle extensions with new attributes.
- Grammar support is more extensive for (child) elements than attributes.
- Since there may be default attributes added by the XML processor at processing, all information may not be available in the XML application instance thus lowering readability and understandability.
- More than one attribute can be used for an element, however, this does not impose any structure.

We could express a general rule as:

The start approach is to use child elements. If there are convincing arguments encouraging attributes use them instead. If there are no convincing arguments (or none at all) stick to child elements.

The following types of attributes exist:

- Enumerated i.e. the possible values are listed (tokens)
- String (CDATA) i.e. not parsed text
- ID, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS

A specific functionality is possible using the ID and IDREF attribute types (IDREFS is used for referencing more than one). Following the example above we can have an XML application instance that contains several vehicles each using the vehicle element. By defining an ID attribute value for each vehicle element it is possible to enable handling of specific vehicles among all those defined. Thus the important thing is that the ID value is unique and this can be checked by the parser.

PCDATA is used for text declarations showing that the text shall be parsed. A CDATA section contains text that is not parsed. Thus it is possible to transfer e.g. an XML application instance embedded as plain text within a CDATA section without the parser checking it.

A *processing instruction* is used for passing information directly to the application how to handle a specific item. The parser just passes this information without affecting it. A processing instruction is written as:

```
<?name instruction ?>
```

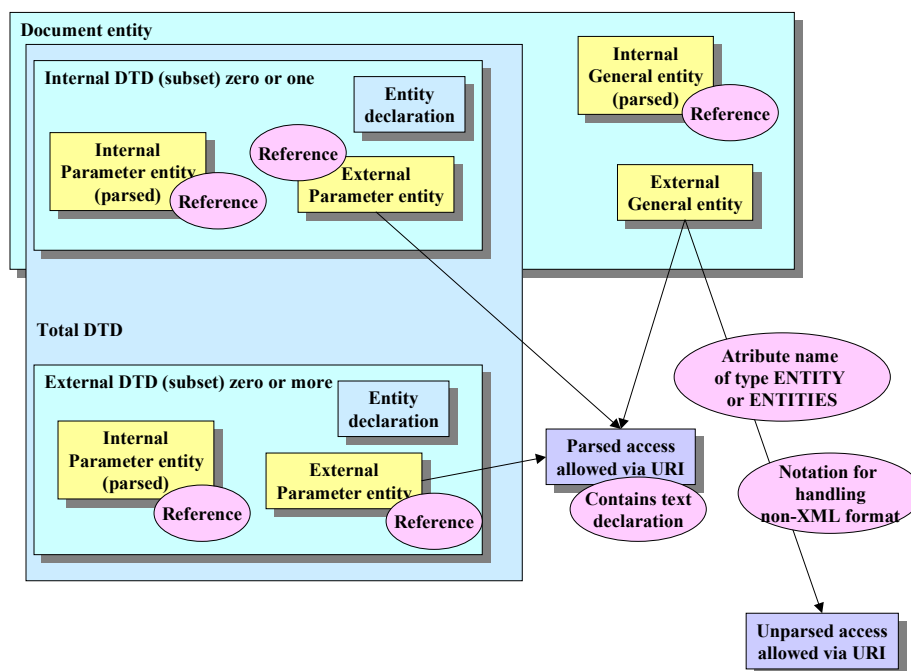
For example, we see that the XML declaration is in principle a processing instruction where *xml* is the name and *version="1.0"* is the instruction.

A *notation* is used for explaining, or give further information to, elements and attributes. A notation is used for one of the following reasons:

- to describe how to handle an unparsed entity (e.g. binary data, see below)
- for an element (non-empty) using a notation attribute i.e. an element that shall be handled in a specific way
- for a processing instruction i.e. to connect extra processing information with the handling of it

4.5 Physical structure

The physical structure describes how different pieces of the XML application instance are physically distributed e.g. among different servers on the Internet. The figure below gives an overview where an ellipse contains a comment.



These pieces (named storage units in the XML specification [6]) are called *entities* and cause some confusion because their definitions are not completely obvious. An entity is used for content and must be declared. We give a summary for clarifying matters. First we consider the “consist of”-aspect:

- An XML application instance *consists* of entities, the DTD does not
- The top entity of an XML application instance is the document entity (or root)
- An entity can consist of other entities
- An entity can be internal or external, if it is internal it belongs to the storage unit where it is declared
- An entity could contain parsed data or unparsed data

Next we consider the “reference”-aspect:

- The % character is used for referencing an entity (external/internal, parsed/unparsed) in a DTD, the & character is used in the same way for an XML application instance
- An entity can reference other entities

- There are predefined entities accessible via the & character in the XML application instance. These are considered external since they do not actually belong to the XML application instance

For example, the root is an entity since the whole content could be stored in the XML application instance itself. An entity could be *internal* i.e. completely defined within the instance. If not, the entity is *external*. One such example is an image file referenced by a URL. An internal entity is always plain text but the external one could also contain binary data.

Irrespectively of the definition of entities (which is somewhat inconsistent) the use of them is the important thing and easier to understand: an *entity* is a named piece of information that can be referenced by its name. When a parser detects an entity reference name the parser replaces it with the content of the entity. There are several rules for this that must be obeyed by an XML processor.

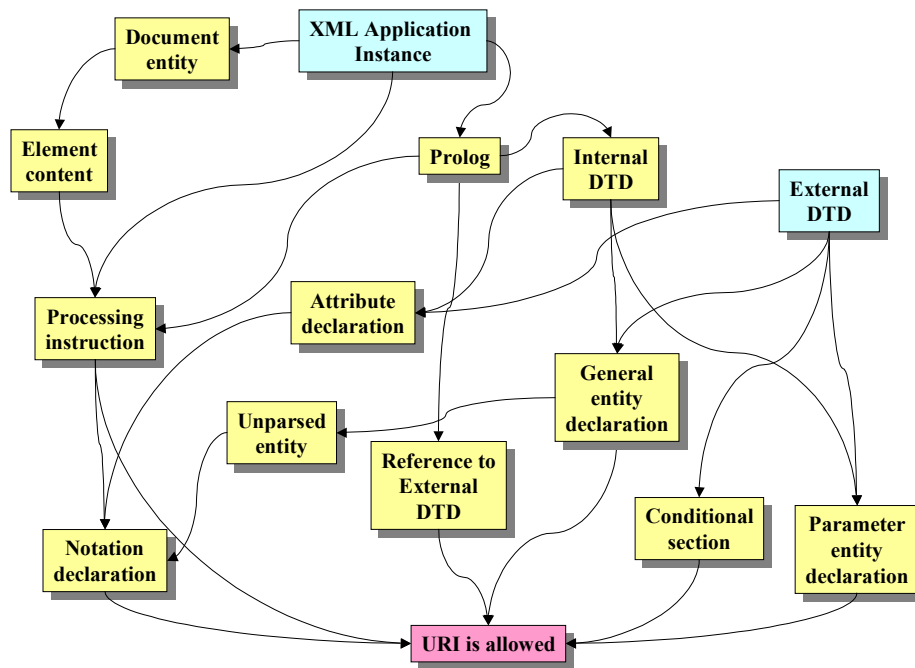
Parameter entity references can only occur inside the DTD. For example, a text that occurs at several places in the DTD could be replaced by a parameter entity which is referenced wherever the text is needed. Thus the text is specified just once and, further, will automatically give an updated document when modified. General entity references are used within the XML application instance. There are a few predefined entities (like & and <) and it is possible to create new ones by declaring them in the DTD.

External entities is a means for making information available to other users and also to compose an XML application instance from a selected set of constituents.

4.6 Use of URI

A URI is a general address and includes the possibility of external access. The picture below shows the ways of accessing external items via URI as defined by the XML specification (see [6]). If external items are used new aspects have to be handled:

- *version handling* – Which external items can be used together? When modified are they backward compatible?
- *dependences* – Are the external items available for others? What dependences are then created? Is an external item a standard?
- *use of externally created external items* – Could they be trusted? How are they updated and version handled? Are they always available?
- *structure* – Is the structure of external items understandable? Is it maintainable?



We see that many possibilities exist and thus it is important to keep track of the corresponding structures.

4.7 Naming conventions

The XML standard (see [6]) contains complete information on how to create valid names for markup information such as element and attribute names. There are some specific rules but nothing that really limits the naming. However, note that names are case sensitive and that “xml” and combinations of upper and lower case characters of x, m, and l such as XML, xml etc are reserved.

The most important aspect, as in a software program, is to find names that really express the intention, are readable and understandable by humans with little prior knowledge. Some characters should be avoided since they could be misinterpreted e.g.

- Arithmetic operators e.g. “-“ in A-B could unintentionally be interpreted as the difference.
- Period “.” e.g. in A.B could unintentionally be interpreted as showing a structure with A belonging to B or vice versa.

Since naming convention can be closely related to the area of interest it is necessary to create a document giving the rules for those aspects that are not handled in the XML standard. Some aspects are the use of underscore, lower/upper cases e.g. which is preferred of LOCAL_NAME, Local_Name, local_name, LocalName?

4.8 Namespace

Namespaces are just mentioned in the XML specification [6]. A specific namespace specification is given in [15] where both version 1.0 and 1.1 are included. Version 1.0 is the generally accepted one but version 1.1 is an updated document, however draft, with one significant addition; the possibility of undeclaring prefixes (this is described below). A

processor that supports a specific namespace can perform specific actions when the namespace is found in the XML application instance.

A namespace relates a collection of names that can be used *only for attributes and element names*. By associating a prefix with the collection it is possible to address specific names in the collection via “prefix:name”. An example shows the basic idea:

```
<vehicle xmlns:car="http://www.xxx.yyy">
  <car:speed>100</car:speed>
</vehicle>
```

xmlns is a predefined attribute that is used for namespace declarations. The declaration looks like an attribute, however, it is not really treated as such. The prefix “car” is associated with the collection of names named by the URL `http://www.xxx.yyy`. *Note that the URL does not have to point to anything, it is just a string to identify the collection!* One of the names in the collection (for an element in this case) is “speed” which can be used via the defined prefix i.e. “car:speed”.

It is possible to set a default namespace. The example below is the same as above but with default namespace.

```
<vehicle xmlns="http://www.xxx.yyy">
  <speed>100</speed>
</vehicle>
```

The declaration of a namespace can only be made within an element. An important aspect is the scope of a namespace. In the example above the scope is within the vehicle element (from, and including, the start tag and element content up to the end tag) but not outside it. Child element could also declare new namespaces or set defaults and their scopes would be parts of the overall vehicle scope thus following normal scope rules. A default namespace can be set to undefined. For version 1.1 of XML namespaces it is possible to also undeclare prefix.

Note that there is a difference in scoping between attributes and element names. The namespaces of attributes and element names are separate. Further, for each element type there is a separate namespace for its attributes. Thus a default namespace for an element does not apply to an included attribute without prefix.

Also note that an attribute or element name may not belong to any namespace having a null URI.

We see that use of namespaces makes it possible to:

- separate elements with the same name but belonging to different namespaces
- make names more generally available and accepted i.e. to create some kind of standard
- to use already defined names i.e. to adhere to some kind of standard
- to override names e.g. in order to allow more/new child element
- to hide prefixes by using defaults
- to redefine already used prefix and to set default namespace

However there is a risk of using prefix instead of attribute for including semantic information i.e. to not treat the prefix as a name but instead treat it as an extra source of information.

As a summary, it is possible to create XML application instances with too complex structures and also too difficult to read. On the other hand just using the basics of namespaces in a restricted manner will improve readability and not cause any problems. In any case, for an application it is necessary to create a document defining rules and guidelines which must be obeyed by all people involved.

4.9 Reserved attributes

There are special predefined attributes that could be used. These have to be declared before use.

- `xml:space` indicates that white space should be preserved
- `xml:lang` indicates the language used in contents and attribute values
- `xml:base` gives the base URI that could be further specified by adding relative addresses
- `xmlns` is used for declaring a namespace

The `xml` prefix (in the first three lines above) belongs to the default namespace for any XML application instances i.e. <http://www.w3.org/XML/1998/namespace>. The predefined attributes are administrated by the W3C organisation.

4.10 XML Infoset

The XML Infoset (see [28]) defines an abstract data set applicable for a well-formed, but not necessarily valid, XML application instance. The rules for namespaces must be followed and relative URI addresses in namespaces are not allowed. XML Base is included as a normative reference.

The XML Infoset defines a number of *information items* possibly created after parsing an XML application instance. Each information item (in italics below) contains a number of properties (within brackets):

- *Document* - [children] [document element] [notations] [unparsed entities] [base URI] [character encoding scheme] [standalone] [version] [all declarations processed]
- *Element* - [namespace name] [local name] [prefix] [children] [attributes] [namespace attributes] [in-scope namespaces] [base URI] [parent]
- *Attribute* - [namespace name] [local name] [prefix] [normalized value] [specified] [attribute type] [references] [owner element]
- *Processing Instruction* - [target] [content] [base URI] [notation] [parent]
- *Unexpanded Entity Reference* - [name] [system identifier] [public identifier] [declaration base URI] [parent]
- *Character* - [character code] [element content whitespace] [parent]
- *Comment* - [content] [parent]
- *Document Type Declaration* - [system identifier] [public identifier] [children] [parent]
- *Unparsed Entity* - [name] [system identifier] [public identifier] [declaration base URI] [notation name] [notation]
- *Notation* - [name] [system identifier] [public identifier] [declaration base URI]
- *Namespace* - [prefix] [namespace name]

The bracket notation is commonly used e.g. in recommendations where conformance to the XML Infoset must be stated when discussing processing and parsing. Thus the use of the XML Infoset is for most persons more of a theoretical interest.

4.11 Metadata

If there is a need for expressing data about data, i.e. metadata, using URI there are some initiatives today that should be considered. One example is the Dublin Core Metadata Initiative (<http://dublincore.org/documents/dcmi-terms/>) as described earlier.

4.12 Conclusions

This chapter has provided a short glance of XML showing the most important constituents. However, grammars and support around XML have not really been considered. Also presentation of data has not been discussed. The nice thing is that data contents and presentation are separated in XML (opposed to HTML) and thus it is possible to use more than one presentation layout without changing the XML application instance as such i.e. without changing the data source.

We see that there are just a few things to keep in mind before one actually could start creating an XML application instance. You get a long way just by considering elements and attributes! But for creating an XML application in a professional way and possibly as a standard this simplicity is a kind of illusion. It might be so that the final XML application proposal is just as simple as the initial idea but there is a very important difference: designing the XML application professionally will raise a number of questions that have to be answered before continuing while just relying on the initial idea cannot be motivated.

5 XML support

5.1 Introduction

Below is a number of support functions described that could be used in order to increase the value and usefulness of XML. Important to note is that they are more or less orthogonal to each other so more than one could be applied at the same time. The support is formally specified in standards and recommendations.

If there is a need for support functionality it is better to stick to a defined standard than to invent something new. Some of the reasons are:

- Following a standard will make acceptance of a new XML application much easier.
- The standard as such will be strengthened.
- There is or will be tool support.
- You save time and money.

Under very special circumstances it might anyhow be necessary to create your own functional support but probably enough functionality could be found in established standards and recommendations since there are so many. It may also be possible that your interest is just within a subset of an established standard but it is still better to use it instead of reinventing the wheel.

5.2 Processor

A useful notion is the (logical) processor. One example was presented earlier, the XML processor, which parses the XML application instance in order to check for errors etc. For the support functions described here there are associated processors. The general tasks for them are to check the syntax and signal errors.

5.3 Relative resource addressing

5.3.1 XML Base

The XML Base recommendation (see [19]) consists of just one attribute; the `xml:base` where `xml` is bound to the namespace <http://www.w3.org/XML/1998/namespace>. The purpose is to define a URI as a base address and use relative addresses to the base address when accessing resources. We get the following properties:

- The XML application instance gets smaller since the base address does not have to be repeated everywhere, one place is enough.
- It is easy to change if a new base address shall be used or if a further refinement shall be made (could be made relative the current base).
- One must know the scope rules since the base address is not visible (apart from the declaration). Further, any child element can declare a base address and thus normal scope rules apply.
- One must know what is the base address e.g. when `xml:base` is not specified.
- XML Base is a standalone recommendation but is a required recommendation for other XML support (see below).

The applicability of XML Base for some other standards is described in Appendix C (non-normative) of [19].

The associated processor tasks are to check for errors and add the correct base address to each relative address.

5.4 Addressing fragments of resources

5.4.1 XPath

XPath is defined for making it possible to select parts of an XML application instance (however not the XML and DOCTYPE declaration parts). By using XPath expressions in a URI it is possible to address pieces of information from an external resource. Another use is to include XPath expressions in attribute values. What to do with the pieces is outside the scope of XPath, *XPath is only a part of a wider context*. One example could be to merge specific parts with a master document.

XPath considers the XML application instance as a tree of *nodes*. The following node types exist:

- *Root node*
- *Element node*
- *Text node*
- *Attribute node*
- *Namespace node*
- *Processing instruction node*
- *Comment node*

The nodes are in accordance with the XML specification (see [6]). There are several rules in XPath relating different types of nodes e.g. “elements never share attribute nodes”. Note that XPath uses a non-XML syntax and thus has to be included as strings.

The advantage of using XPath is that information of the resource does not have to be tagged, instead the structure of XML application instances can be utilized in order to address pieces of information. XPath addresses whole nodes but XPointer (see below) can address parts of nodes and also parts that cross node borders. XPath can not be used for a not well-formed document such as an external parsed entity.

An expression in XPath is one of the following types:

- a set of nodes
- a boolean value
- a number
- a string

Variables of these types can be defined. An actual implementation of XPath also contains a number of predefined functions with input arguments and results according to these types. Functions are parts of expressions and a function example is conversion from one type to another.

For the XPath functionality the most fundamental aspect is the *Location path*. It is a kind of expression and selects a specific set of nodes e.g. some of the children elements of a specific parent element, all attributes of a specific node etc. A location path consists of a number of *location steps* separated by / such as:

```
child::doc/child::chapter/child::headline
```

where:

- the first location step selects all the “doc” elements (relative the current node)
- the second location step selects all the “chapter” elements of all the “doc” elements
- the third location step selects all the “headline” element of all the “chapter” elements of all the “doc” elements

A location step can contain three parts:

- *Axis*. It is used for specifying the path when considering the XML application instance as a tree of nodes (according to above). Some examples are: child, parent, ancestor.
- *Node test*. Some examples are: a specific element, all attributes.
- *Predicates (zero or more)*. An example is: element number 3 in a list of nodes.

There is also a defined abbreviated syntax in order to simplify the rather verbose character of XPath e.g. @ stands for “attribute:”. On the other hand the abbreviated syntax is, to some extent, in conflict with the general intention of XML as such to be more explicit than e.g. HTML.

An important detail is the handling of arithmetic operators. For example, a minus sign must have a preceding whitespace in order to separate subtraction from a character in a name.

The associated processor task is to check for errors.

As a summary XPath should be seen as a means for other use primarily by XPointer and XSLT (see below).

5.4.2 XPointer

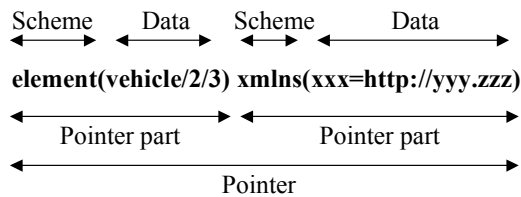
XPointer is based on XPath and uses most of its notation. However, XPointer extends XPath and thus creates new notations as well e.g. *location set* in XPointer extends *node-set* in XPath in order to allow access of fragments of nodes. XPointer uses a non-XML syntax and thus has to be included as strings. XPointer could be used also for a not well-formed document such as an external parsed entity. The recommendation (see [22]) is currently split in four parts:

- Framework
- Element Scheme
- Xmlns Scheme
- Xpointer Scheme

matching the defined schemes (see below).

In the same way as for XPath the advantage is that information of the resource does not have to be tagged, instead the structure of XML application instances can be utilized in order to address pieces of information. XPointer can address parts of nodes and also parts that cross node borders.

The basic item is the scheme-based pointer which consists of one or more *pointer parts* separated by white space as shown in the example below.



A *scheme* is defined as a specific pointer data format. If one part succeeds (evaluated left to right), i.e. gives subresources, the remaining pointer parts will not be evaluated. The following schemes exist:

- `element (...)` scheme – to allow basic addressing of elements. This scheme does not support namespace.
- `xmlns (...)` scheme – to define namespaces
- `xpointer (...)` scheme - to address pieces of information i.e. the full addressing capability of XPointer

Note the use of parenthesis enclosing specific information for each scheme. Using the element scheme one first specifies a name giving the start element and then uses number for addressing child nodes. A “/” is used for next child level e.g.

```
element(vehicle/2/3)
```

specifies the third child element of the second child element of the vehicle element.

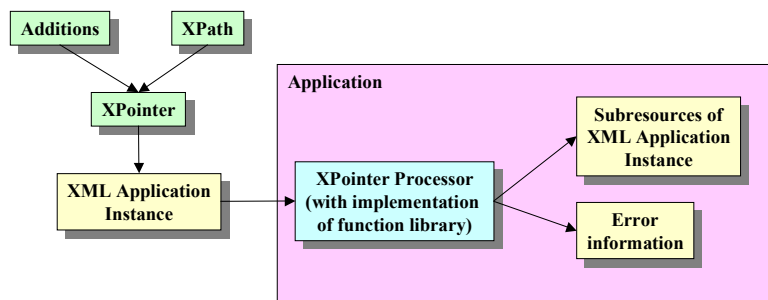
Using the xmlns scheme makes it possible to use namespaces for the remaining pointer parts e.g. defining

```
xmlns(xxx=http://yyy.zzz)
```

makes it possible to use xxx as a prefix i.e. to represent a namespace. Since an xmlns scheme does not identify subresources the evaluation always continues with the next pointer part.

Using the xpointer scheme makes it possible to address pieces of XML application instances. This scheme is a generalisation of XPath e.g. two new nodes *point* and *range* are added (called locations in XPointer instead of nodes in XPath). A point could e.g. be a position between two nodes and a range contains start and end points. XPath cannot address an external parsed entity instead XPointer has to be used. Also a number of functions are added compared to XPath.

The figure below shows the principle view of addressing specific parts of an XML application instance.



The associated processor task is to check for errors and addresses the pieces of information to be used by the application.

As a summary XPointer is more sophisticated than XPath. XPointer should be seen as a means for other use e.g. by XLink, XInclude, RDF, SOAP (see below).

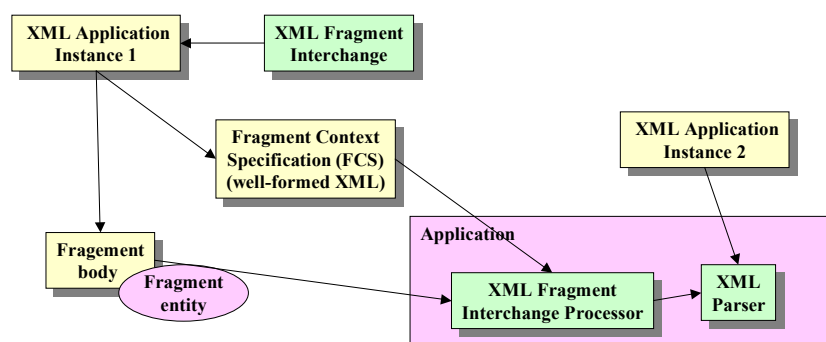
5.5 Parsing fragments of resources

5.5.1 XML Fragment interchange

The recommendation is found in [25]. The idea is to make it possible to parse fragments of XML application instances instead of parsing complete XML application instances in order to find the interesting part. The holder of the XML application instance specifies the necessary fragments which can then be used by others.

A fragment is removed from its (parsing) context and thus it is necessary to transform context information (Fragment Context Specification or FCS) together with the fragment to be parsed (fragment body) concerning e.g. attributes, child elements. Within the FCS a reference to the fragment body is made using the *fragbody* element (specific for the namespace used for XML Fragment interchange).

For using the fragment a reference is necessary to the FCS which in its turn contains a references to the fragment body. The picture below shows the principle where an ellipse contains a comment.



This recommendation lists and describes possible parts of the fragment context specification but does not state what is mandatory. This depends on the actual situation and has to be

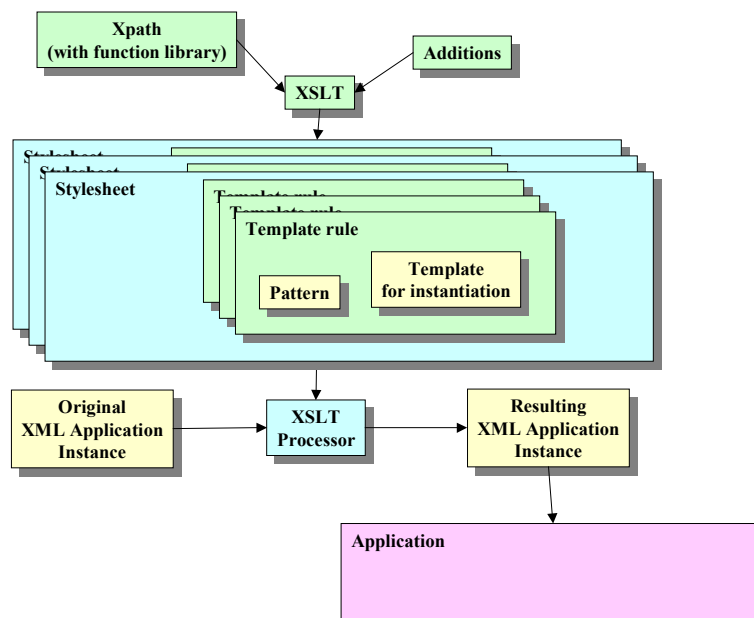
specified between sender and recipient of each fragment. Thus the recommendation should be seen as a framework.

The associated processor task is to check for errors and include the fragment body into the FCS before parsing.

5.6 Transform XML application instances

5.6.1 XSLT (eXtensible Stylesheet Language Transformations)

XSLT is a language for transforming an XML application instance into something else. XSLT is an XML application and is an extensive recommendation. XSLT is specifically suited for XSL (i.e. for presentation, see below) and for that reason a transformation is generally called a *stylesheet* (also *transform* occurs sometimes). A specific namespace is defined for XSLT but only used in the stylesheet (i.e. not in the XML application instance). Only the stylesheet declaration is in the instance.



As shown in the picture XSLT includes and extends XPath. For handling transforms there are two main aspects:

- *pattern* – used for finding information to be transformed in the original instance (source tree representation). Pattern uses XPath expressions.
- *template* – instantiated when creating information in the resulting instance (result tree representation)

More than one template could be applicable and priority rules are defined for handling conflicts. For patterns, a subset of XPath expressions is used.

Note that it is possible to use more than one stylesheet by using *inclusion* and *import*. This makes it possible to override and add template rules. However, there is a risk of making too complex solutions when using more than one stylesheet e.g. by introducing complex dependences.

By using a *mode* attribute it is possible to process an element with different result depending on its mode value. There are many constructs included, some examples are:

- create literal element i.e. a copy from source tree to result tree
- create new element
- create new attribute
- create attribute set (can be nested)
- create text
- create processing instruction
- create comment
- copying node
- repetition
- conditional processing if-then, one of several alternatives
- sorting
- variable and parameter
- functions
- message
- fall back
- output
- named templates

We see that the capability resembles a programming language with limited structural support but with extensive support for manipulation of the source XML application instance. An important aspect is then to comment the stylesheet thoroughly in order to make it readable.

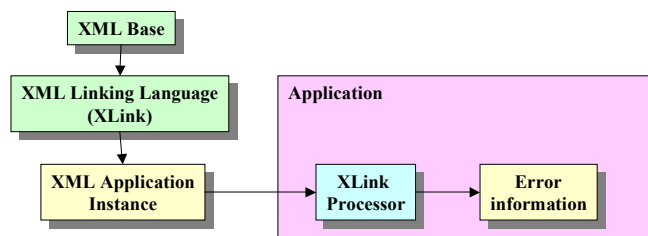
Apart from use within XSL an important use of XSLT is to handle interfaces between instances of similar or co-operating XML applications.

The associated processor task is to check for errors and produce the result tree.

5.7 Structuring resources

5.7.1 XLink

The idea is to make it possible to create links to resources and to describe relations between them thus creating resource structures. The recommendation is found in [20]. The defined namespace is <http://www.w3.org/1999/xlink/>. The figure below shows the principle relations.



XML Base is included as a normative reference and XPointer may be used for references but is not a requirement.

XLink does not specify any elements instead it defines a number of attributes that can be used for elements. XLink specifies two types of links:

- *simple link* between two resources (like hyperlink in HTML) including no internal structure.
- *extended link* that is a general link between any number of resources and also including internal structure.

Using simple link makes it possible to:

- To access a single remote resource.
- To access a local resource (inside the link element)

Using extended link makes it possible to:

- Associate many resources that are in some way related however not obvious when considering each alone.
- Describe links using labels and roles.
- Describe how links are connected (from, to).
- Give rules for how to traverse links.
- Access a local resource (inside the link element).
- Reference a linkbase i.e. a collection of externally emanating links that fulfil XML rules.

The basic XLink attribute is *type* which is used for classifying the element containing it. Once classified, different additional attributes could be included in order to further specify the element. The following values are possible for the *type* attribute:

- *simple* – specifies a simple link
- *extended* – specifies an extended link
- *locator* – address to remote resource
- *arc* – rule for traversal between resources
- *resource* – specifies a local resource
- *title* – label (for humans)
- *none*

and, as said above, specifies the element containing the *type* attribute. The two first are special, from a logical point of view, since they specify the possible types of links; simple or extended.

There are also other attributes:

- The *href* attribute is used for certain values of the *type* attribute and specifies a URI to a remote resource.
- The *role* attribute is used for certain values of the *type* attribute and specifies a URI to a resource describing the intended property of the whole link (i.e. all relationship between resources).
- The *arcrole* attribute is used for certain values of the *type* attribute and specifies a URI to a resource describing the intended property of the arc.
- The *title* attribute is used for certain values of the *type* attribute and contains a string describing the meaning of a link or resource.
- The *show* attribute is used for certain values of the *type* attribute and defines behaviour intentions to the ending resource. The following values are possible: new, replace, embed, other, none.

- The *actuate* attribute is used for certain values of the *type* attribute and timing. The following values are possible: onload, onrequest, other, none.
- The *from* and *to* attributes can only be used for the *arc* value of the *type* attribute and specifies start and end points.

Default values are allowed making the description less verbose but putting higher requirements on the knowledge of the reader. Note that URIs, e.g. web addresses, could be used not only for describing link start and end points but also for descriptions. Thus external structures could be built up concerning the meaning of link structures.

There are many rules for required, optional and not allowed combinations of attributes, attribute values and elements classified by the attributes. This makes the situation complex and the rules have to be studied carefully. The notation is not intuitive e.g. the *title* attribute has double semantics and the special meaning of *simple* and *extended* are not separated from the other type attributes. However XLink could be of great value if much dependence is put on (external) resources. The question is of course if these must/should be used in the first place, however, this is strongly application dependent.

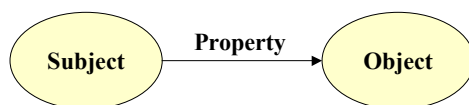
The associated processor task is to check for errors.

5.7.2 RDF

RDF stands for Resource Description Framework (see [33]) and, among other things, defines a specific XML application called RDF/XML. RDF/XML supports XML Base. A good start is to read the RDF Primer (at <http://www.w3.org/TR/rdf-primer/>) and thereafter RDF Concepts and Abstract Syntax (at <http://www.w3.org/TR/rdf-concepts/>).

RDF is defined for making it possible to give statements about resources via extensive use of URIs and thus making it possible to e.g. structure information available on the web. A specific namespace is associated with RDF. Note that RDF is a framework and cannot e.g. be used for evaluating correctness of resources.

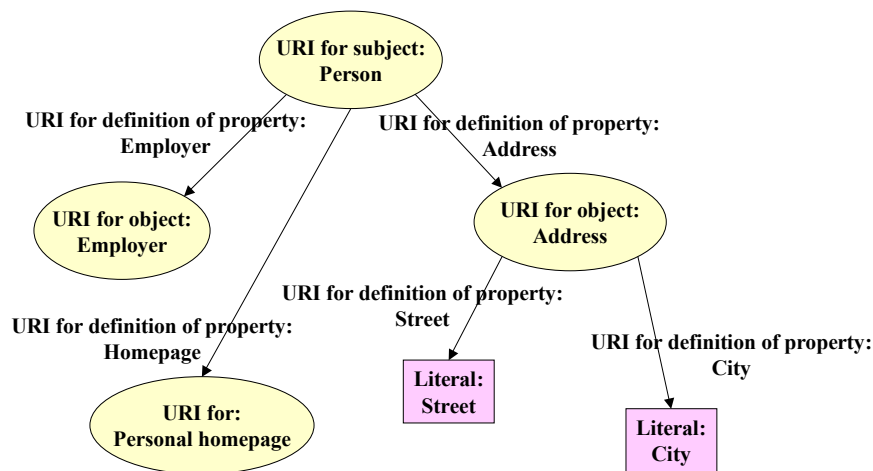
The figure shows the basic idea.



Thus RDF uses:

- *Subject node* – the item to give statements for e.g. John
- *Property (predicate) arc* – the property of the subject e.g. owns
- *Object node* – the value of the property e.g. a car

The property arc represents a statement concerning the relationship between Subject and Object. The most important thing is that all three constituents use URIs. The example below shows this.



A literal denotes a text string. An object can be a subject in another part and also a property can be a resource.

The RDF/XML is defined for making it possible to serialize the graph i.e. for creating a text document with the same information as the graph. Since a graph can be very complex there is support in RDF for grouping things that belong together in one way or another:

- *bag* – an example of a *container* item. It is an unordered group of resources or literals.
- *seq* – an example of a *container* item. It is an ordered group of resources or literals.
- *alt* – an example of a *container* item. It is a group of alternatives resources or literals where one of them can be selected.
- *collection* – a closed list of resources or literals i.e. all members are known (in contrast to container).

RDF also allows the possibility of giving statements about statements e.g. to specify when and by whom a statement was made.

RDF Schema defines a framework for classes (e.g. concerning subclasses, inheritance), property descriptions, description of RDF vocabulary and other aspects.

The associated processor task is to check for syntactic errors.

5.7.3 Conclusions

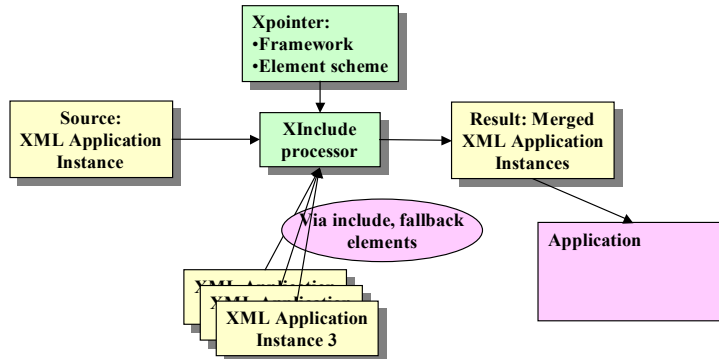
Both RDF and XLink are defined for making it possible to structure resources, however, they start from different perspectives. XLink focuses on how to specify the links as such and RDF focuses on relations between items. Another way to see it is that RDF takes a top-down perspective “what resources do we want to model” and XLink takes a bottom-up perspective “how do we connect to resources”.

As a general precaution one should first consider the need for (complex) resource structures since there might be a considerable effort to maintain and extend the structure in the future especially concerning inconsistencies and availability of resources.

5.8 Merging XML application instances

5.8.1 XInclude

The idea is to actually merge different XML application instances into a new one as shown in the picture below where an ellipse contains a comment.



Note that XInclude is independent, among other things, of parsing and grammars. XInclude should instead be seen as a preprocessor. XInclude has an associated namespace <http://www.w3.org/2001/XInclude> and supports XML Base. XInclude defines only two elements (with associated attributes):

- *include* – the basic means for inclusion.
- *fallback* – the fallback element is a child element of include. If include fails the fallback element is processed instead.

The included information is either parsable XML or plain text. XInclude makes it possible to keep different XML application instances separate and including them when needed. However, the question is to what extent such structures are practical and safe in a specific application. Two important aspects are version handling and availability.

The following attributes are defined for the *include* element:

- *href* – this could be an XPointer and is an URI to the resource to be included. Note that only the XPointer framework and the XPointer element scheme are required but XPointer xpointer scheme may be included.
- *parse* – has the value “xml” if the resource is parsable XML otherwise the value “text” i.e. treated as a text
- *encoding* – only used for *parse* value “text” and defines the encoding of the text

Nesting is allowed but not those causing e.g. infinite loops due to recursion. Special handling is needed when IDs are used in the included part e.g. there could be collisions with already used IDs. Also there could be problems with use of namespaces when the included part assumes a specific namespace not used in the source XML application instance.

The associated processor task is to check for errors and merge the source XML application instance with included items into a resulting XML application instance.

In a way, external references and XInclude address the same issue. However, XInclude deals with already created XML application instances as opposed to external references which are used at parsing.

5.9 Presentation

5.9.1 Introduction

As described earlier there is no information of how to present an XML application instance. Instead information has to be given in order to present data e.g. on a web interface using HTML.

There are a few possibilities to choose between:

- To use CSS (see below). The principle is to directly map elements defined in the XML application with how they shall be presented.
- To use XSLT for transforming the XML application instance into HTML, XHTML or another version of XML and to use CSS for the result.
- To use XSLT for transforming the XML application instance so it can be used with XSL (see below).

Two important things to keep in mind are:

- Keep things simple.
- Is a complex transformation really necessary?

5.9.2 CSS Cascading Style Sheet

The CSS recommendation is found in [29]. Even if originally defined for HTML-documents CSS can be applied to XML application instances as well. “Cascading” stands for the possibility of using more than one style sheet for presentation. This makes it possible to use inheritance, merge styles and to override styles based on priority. Note that it is possible to set priority explicitly. There are a number of rules defined for handling priority issues especially at conflicts.

By using CSS it is possible to connect style to a specific element e.g. deciding font, size etc for the element <vehicle>. If different elements shall have the same style they can be grouped together and child elements can inherit their parents.

To attach style individually e.g. for different instances of the same element there are the following possibilities:

- by using *class*. An element can have an attribute class with a specific name as the value. A style could be attached for all elements with the same class name.
- by using the ID attribute (which must be unique). Style could be attached individually for each ID.
- by using a search pattern i.e. by applying a number of simple selectors in a sequence. Examples of simple selectors are child and parent.

Note that these possibilities might blur the overview and structure of the presentation since several more or less independent structures could be created.

CSS allows presentation to be affected by:

- the author
- the presentation unit e.g. a web browser showing different colours for a link that has been traversed and not

- the reader which might add personal preferences to the presentation e.g. increased font size

The associated processor task is to perform presentation by using a CSS and an XML application instance.

5.9.3 XSL Extensible Style Language

XSL is based on DSSSL (Document Style and Semantics Specification Language) for SGML documents. XSL-FO (XSL Formatting Objects) is a sub-language of XSL and defined in [32]. An XSL-FO document is also an XML application instance. XSL is an extensive standard about 500 pages long. A specific namespace is defined for XSL.

XSL includes normatively XSLT. The purpose of XSLT (for XSL) is to transform an XML application instance into a format that can be used by XSL (XSL-FO) for presentation. The source tree (the XML application instance) is converted into a result tree which is used by the formatter for presentation. Formatting could contain page number, headers, footers and be adapted for printers, handheld devices etc.

The associated processor task is to perform presentation by using an XSLT style sheet, XSL-FO and an XML application instance.

5.9.4 Conclusions

Since there are two major choices, which is the best to choose? W3C gives the following very reasonable recommendation:

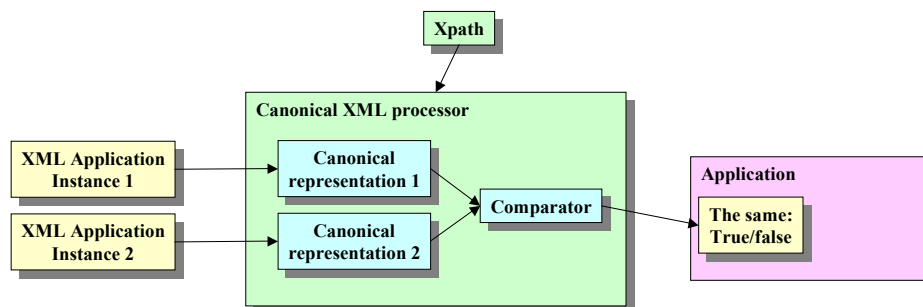
Use CSS when you can, use XSL when you must.

The main reason is that CSS is simpler to learn and use and fulfils your needs in most cases. Also tool support is more commonly available for CSS than for XSL.

5.10 Comparing XML application instances

5.10.1 Canonical XML

The recommendation is found in [26]. The picture below shows the principle behaviour.



The idea is to create the canonical form of two instances and then to compare them in order to find out if they are logically the same even if they are not necessarily identical. In this way it

is possible to verify if they have the same behaviour or if the first version when modified results in a new version that is not logically the same. A further use could be to verify if unauthorised modifications have been made. The canonical form is based on XPath.

The canonical form is created by removing some specific items, removing redundant information, and rectifying others and also making as much as possible explicit. It is *not certain* that the canonical representation can be used instead of the XML application instance. One example of this is that external unparsed entities are not considered.

One must be very careful when interpreting the result. One thing that the canonical form cannot address is equivalent logical behaviour built in, in the application; the canonical form has of course no understanding of the application as such. Also there are 14 allowed modifications when creating the canonical form and thus could imply significant loss of information.

The associated processor task is to check for errors, create the canonical representation and perform the comparison.

As a summary the canonical form should be used with care and under well defined conditions. Two examples are encryption and digital signatures where e.g. an extra whitespace is insignificant but anyhow effects results. To make a comparison significant there must be several explicit and well defined rules for the XML application and its instances as such but also for limiting the degrees of freedom allowed.

5.11 Script languages

An alternative to the support functions described above is to create specific support using a script that is programmed in a specific script language (of course ordinary programming languages could also be used but are not considered here). For example, the script language Perl is especially suitable for texts and thus XML application instances could directly be used as input to Perl scripts. Other script languages are more specific e.g. XML Script (see www.xmlscript.org) which is a direct alternative to XSLT. Some other script languages are Jscript, Python, Tcl, JavaScript, ECMAScript, ASP and VBScript. A further use of script languages is to include code in CDATA sections i.e. where text is not parsed. The user of the XML application instance could then localize and execute the code.

The advantage of this kind of support is that special and optimised functionality can be created. The drawback is that there is yet another program to develop, document and maintain and that it is not a standard. Just like any other program it is difficult to predict future extensions and the effects on the structure of the specialised program.

5.12 Conclusions

From above we see that there are many possibilities for extending the functionality of XML applications. If this kind of functionality is needed there is no reason to reinvent the wheel. The principle must be to use as many applicable standards as possible within the scope of interest (this also includes future extensions). The most important advantage using this principle is that your new XML application will be more generally accepted if based on already existing recommendations and standards.

6 Grammar

6.1 Introduction

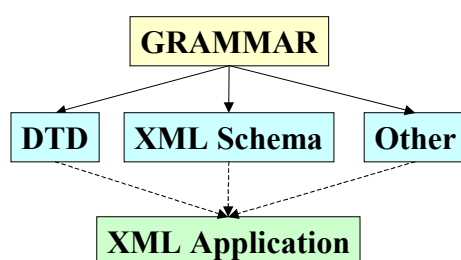
An important aspect of XML is that it is possible to validate a written XML application instance. XML as such enables only well-formedness i.e. only some general rules could be checked e.g. only one top element exists and elements are properly nested. Or put in another way, well-formedness is not unique for an XML application. For validating an XML application instance, a grammar is needed and the start approach is to use DTD (Document Type Definition) at least seen from a historical point of view since support is given directly in the XML specification [6] and is an inheritance from SGML. There are several limitations, however, not severe for limited use. However, there are other, and much better, alternatives as described below.

A grammar contains rules that are used for validating XML application instances against the language defined by the XML application. If all rules are fulfilled the instance is valid otherwise the XML parser will generate error information. A specific grammar is actually the definition of the XML application since it defines rules specifying what is allowed and what is not allowed. The rules concern:

- Element type declaration
- Element content rules including counting: one or more, zero or more etc
- Attribute (or attribute-list) declaration containing the set of attributes for an element type, attribute constraints, attribute default values

Validation makes it also possible to check a limited form of security violation; when receiving an XML application instance and if still valid there is at least an indication of not being manipulated.

There are different, grammar specific, principles for associating an XML application instance with the grammar.



The picture shows a structural overview of possibilities and will be explained below.

6.2 DTD

The DTD is defined in parts in the XML specification (see [6]) but a separate specification of DTD does not exist. There is no structure defined in the DTD only general rules such as an item is declared before being used. Comments will thus play a crucial role for making the DTD understandable. A reference to the DTD is made in the XML Application instance using the XML meta language:

<!DOCTYPE top_element ...>

As mentioned earlier the DTD could be external or included directly in the XML application instance (internal). The following possibilities exist:

- To have a single internal DTD i.e. the grammar is included directly in the XML application instance.
- To have one or more external DTDs i.e. the grammar is defined outside the XML application instance.
- To have a single internal DTD and one or more external DTDs.

In case of conflict and if both internal and external DTDs are used the internal takes precedence. In case of more than one external DTD a DTD following another one, in the order they are listed, takes precedence. Thus there are possibilities for inheritance. Note that an external DTD has no reference to any XML application instance only the opposite association exists. For an external DTD only, conditional sections can be defined that include or exclude parts of the DTD based on a keyword.

The following possibilities exist using a DTD:

- Declare elements and their inner structure; if they only contain child elements or also text, if they are empty, if several choices are available etc.
- Declare rules how many times an element can occur:
 - Exactly once
 - Zero or once i.e. optional
 - Zero or more times
 - At least once
- Declare attributes i.e. names and values, if they are required, default etc.
- Declare entities to be used in the XML application instance (general entities) and only within the DTD (parameter entities).
- Declare notations.
- Include comments.

6.3 XML Schema

An excellent introduction to XML Schema is found in [36]. For the full specification (there are two parts: Structures, Datatypes) see [35]. XML Schema uses a syntax according to XML. The functionality is extended compared to DTD as specified in [6] and only the most important extensions are described below.

An important structuring mechanism is the use of types for elements; simple and complex types exist. Complex types are characterised by:

- Hierarchies of elements i.e. nested elements are possible.
- Can contain subelements and attributes
- Can specify sequence of elements
- Named types that can be extended or restricted in subtypes i.e. inheritance
- Not allowed for attributes
- Can be overridden
- Can be global i.e. allowed in the whole XML Schema or local i.e. the opposite

Simple types are characterised by:

- Not having subelements
- Being strings, integers etc
- Allowed for attributes
- Can be global i.e. allowed in the whole XML Schema or local i.e. the opposite
- New simple types could be derived e.g. restricting an integer type to values within 3-18. Also regular expressions could be used.
- A list type of simple types can be created
- A union type of different simple types can be created
- An element of a simple type cannot have attributes (instead a corresponding complex type has to be created)

Types and elements can be declared as *abstract* i.e. cannot directly be used (this resembles abstract classes in object oriented programming). The use of *abstract* could be to especially point out that further treatment is necessary i.e. the abstract item is some kind of template. Also it is possible to specify that different forms of derived types is not possible for a specific complex type (again resembling object oriented programming).

The occurrence of elements and attributes have more possibilities than DTD e.g. by using min and max values. A *choice* element can be used for specifying that only one child within the choice element can be chosen. By using an *all* element all child elements are optional and can occur in any order. It is also possible to group elements and nest groups and defining the exact sequence of elements. Attributes can also be grouped and nested.

By using groups it is possible to create structures that improve maintenance and usability e.g.

- A group makes it easier to define standards since it could be discussed in isolation
- A group makes it easier to include extensions since all users will be updated at the same time
- A group improves reusability and structure since it encapsulate items that belong together

However, making a group not compatible with the previous versions will affect all users directly. It is also possible to create hierarchies of groups that might result in too complex structures.

XML Schema fully supports namespaces and in this way e.g. multiple XML Schemas are supported. The need for more than one XML Schema can be formulated as:

- A single XML Schema is too long and/or too complicated to overview
- Separating concerns in different XML Schemas makes it easier to define standards
- It is easier to include extensions in separate more limited XML Schemas

Two forms of including external XML Schemas are defined:

- *include* – makes it possible to include
- *redefine* – makes it possible to override

More than one of each could be used and each can in its turn include others. It is also possible to create hierarchies of XML Schemas but this might result in too complex situations.

The ID and IDREF of [6] are extended to allow more complex relations concerning uniqueness of attributes and elements within specific scopes. A subset of XPath is used for this.

An application based on XML Schema can be structured in different ways. The following principal methods can be defined:

- *russian doll* - mirrors the structure of an instance (an element containing subelements that in their turn contain subelements) i.e. local declarations.
- *salami slice* – separates elements and declarations and referencing them when needed.
- *ventian blind* – creates type declaration (instead of creating element declarations in *salami slice*) and using them when creating elements.

Discussion of advantages and disadvantages can e.g. be found in [1] and also mixtures of the principal methods can be used if needed.

From above we see that XML Schema is designed in a controlled, focused and consistent manner and is a mature tool for XML application instances. However, the complete capability of XML Schema might be too extensive to master and too much for normal use. Instead a sufficient subset should be used. The disadvantages compared to DTD is that DTD was defined earlier, is explicitly mentioned in the XML specification and is sometimes sufficient, however not ideal, in small XML applications.

6.4 Alternatives

There is a lot of criticism in the literature concerning both DTD and XML Schema. The former for being insufficient and the latter for trying to be complete but not fully succeeding. For XML Schema there is also criticism concerning complexity and being too big to comprehend. As a consequence new proposals emerge with the general drawback that there will be more alternatives to consider for everyone involved even though they may be better than previous alternatives.

The standpoint taken in this report is that it is not likely that all features will be used in an advanced grammar since they are probably not needed and will make things more difficult than necessary for all people involved. It is assumed that for specific problems there are alternative solutions that might be clumsy, verbose or have other minor drawbacks but maintain the possibility of understanding for non-experts. This principle is encouraged here since it is believed that the quality of documents increases with the number of people understanding them and not being experts.

However, we give some examples below of possible solutions that might succeed in the future. One important aspect if this will happen is the quality and the extent of tools supporting them.

- RELAX (Regular Language description for XML) see <http://www.xml.gr.jp/relax/>
- TREX (Tree Regular Expressions for XML) see <http://www.thaiopensource.com/trex/>
- RELAX NG unifies RELAX and TREX see <http://www.oasis-open.org/committees/relax-ng/spec-20011203.html>
- DDML (Document Definition Markup Language) <http://www.w3.org/TR/NOTE-ddml>
- Schematron <http://www.ascc.net/xml/schematron/>
- SOX (Schema for Object-Oriented XML) <http://www.w3.org/TR/NOTE-SOX/#ref-dcd>

In the list above Schematron is special since it uses assertions that are tested and where the order of assertions is not significant. One example of assertion is: a specific element must

have a single specific child element. XPath and XSLT are used. Schematron uses XSLT for first creating a “meta stylesheet” which is then used for creating the actual stylesheet used for validation (see [39]).

The other examples above are more like DTD and XML Schema in their behaviour.

6.5 Conclusions

At the moment there are two real alternatives: DTD and XML Schema. The rest does not currently have general acceptance. A mix of different grammars is generally not recommended due to increased complexity so one grammar should be selected. The basic aspect is the complexity of the XML application and the general dilemma is:

- XML Schema is much more capable and better structured than DTD, but
- DTD has the advantage of being first and also has the connection to SGML

The list shows the major drawbacks of using DTDs:

- does not have a syntax that is according to XML.
- does not support specifying data types such as digits, date, string
- does not support bounds for acceptable data ranges
- does not support inheritance of schema classes
- does not support namespaces
- does not support complex data types

All these are supported in XML Schema.

An important question is also the allowed freedom built into the grammar. One example is the use of ANY for element content, another is how often an element is allowed according to limits. The question is strongly related to the following aspects:

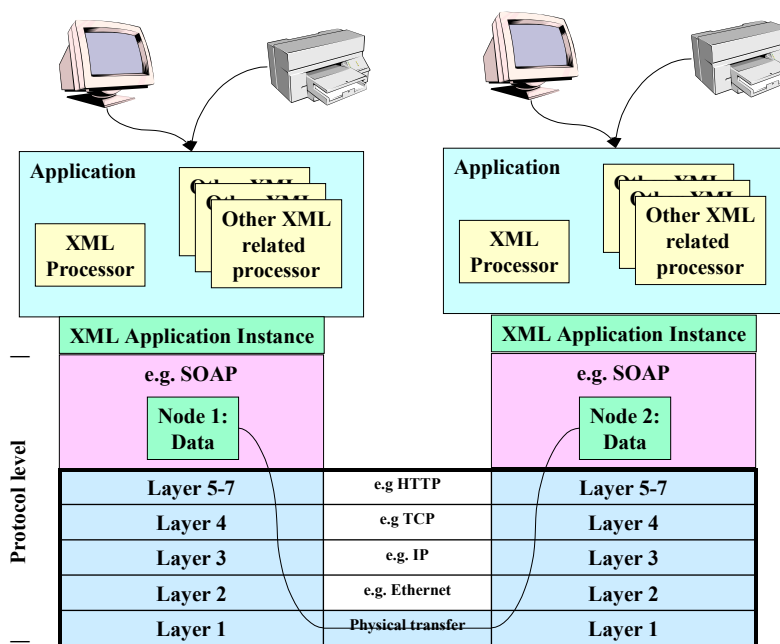
- The need for future modifications and extensions and possibly also restrictions
- The need for accepting different kinds of XML application instances
- How high the knowledge is of the application area
- The amount of consensus among involved parties concerning what to accept and not accept
- The capability of the chosen grammar

7 System aspects

7.1 Introduction

In this chapter we will give general comments concerning principle system design and implementation in relation to XML.

Using XML implies the use in distributed environments and especially use of the Internet. However, the same technology could also be used for Intranet, i.e. “local Internet” not connected to the Internet, and Extranet i.e. “local Intranet” with limited access to the Internet. An example of a complete application is shown below.



As shown in the picture XML application instances are not normally visible to the users of the systems. Instead they are a means for structuring data and to enable communication across different node platforms. Communication takes place using a number of *protocols* which will be discussed below. The XML processor shown in the figure is the parser. There are also a number of other XML related processors implemented dependent on the needs of the application. Generally they will check for errors and perform some specific tasks such as merging different fragments of XML application instances.

We can give some guidelines for the communication of XML application instances:

- The execution overhead in terms of execution time, memory requirements etc shall be low.
- Security aspects are important i.e. integrity, authorisation, authentication, confidentiality and non-repudiation. At which level or levels they should be implemented at needs further considerations.
- COTS product should be used as much as possible since they are cheaper, well tested and generally accepted.

7.2 Transaction

We here define transaction to mean the sequence of transfers of XML application instances belonging to the same specific scenario. An example scenario is “to withdraw money using an automatic teller machine (ATM)”. The scenario could be implemented using a transaction with the following transfers:

- ATM to Bank: Check identity of person.
- Bank to ATM: Confirmation.
- ATM to Bank: Request money.
- Bank to ATM: Accept request and deliver money.

For an application there will be several scenarios and transactions. A transaction could contain different numbers of transfers going in one direction compared to the other. A transfer could go from one to one, one to many, many to one and many to many. It might be so that a transaction can be used for more than one scenario (just by modifying data content) and more than one transaction can be used for a specific scenario. Also there could be transactions having common transfers and/or common subsequences of transfers.

The question now is the granularity of XML application instances. We have two extremes:

- To send all information in a single large XML application instance and let the receiver extract the relevant parts.
- To split information in so many small XML application instances as necessary and then compose the relevant ones before sending to the receiver.

We could of course also land somewhere in between these extremes.

It is not possible to give a general advice but we could list a number of aspects that will affect the decision:

- *execution overhead* – always sending and parsing a big file takes time
- *administration* – a single file gives minimal administration
- *composing pieces* – might require a language on its own
- *separation into pieces* – improves reusability
- *separation into pieces* – moves complexity from a single collection into many small collections and their relationship
- *future development* – what aspects will affect in the foreseeable future?

The general guideline is to at least be aware of this aspect and take a top-down decision i.e. at the beginning of the development project.

7.3 Data transfer

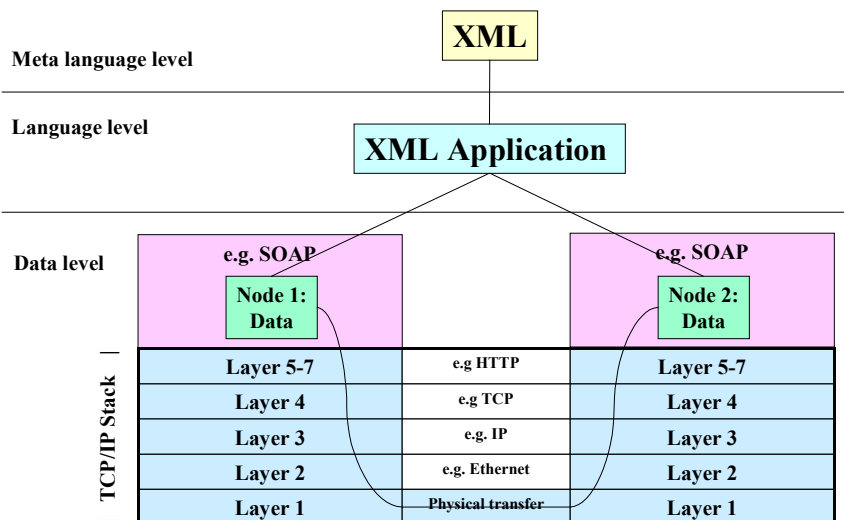
In order to communicate in a distributed environment it is necessary to use standardised *protocols* for handling the actual transfer of data. A sender using a specific protocol adds administrative information to data. The receiver removes and checks the information to verify that the data was transferred correctly. Since there are different protocols with different purposes and granularity it is practical to use a protocol stack i.e. a hierarchy of protocols. The hierarchy is normally classified according to the OSI model, from [4] we have:

Layer 7: Application	This layer handles communication between applications.
Layer 6: Presentation	This layer handles a standardised data representation and assures that sent data could be correctly read.
Layer 5: Session	This layer handles flow control (e.g. not sending more than the receiver could handle) and communication sessions i.e. establishes and terminates them.
Layer 4: Transport	This layer handles connection, data flow control, error handling and assures error free data.
Layer 3: Network	This layer handles connection and data transfer. Logical networks and addresses (IP-addresses) are used.
Layer 2: Link	This layer handles data transfers. Physical network and addresses are used for transmit and receive.
Layer 1: Physical	This is the hardware layer.

We can give some reasons for using protocols and a hierarchy of them:

- If standard protocols are used it is easier to connect to other systems.
- The protocols are thoroughly tested e.g. concerning retransmission, error handling.
- The application does not have to consider the actual transfer of data but could concentrate on the behaviour instead.
- A hierarchy makes it possible to handle specific functions at the lowest possible layer i.e. it is possible to decrease the communication overhead.

The picture below shows the connections between XML, XML application and protocols. The curved line shows the data flow path through the different protocol levels.



For communication via the Internet, layer 5-7 are normally treated as a single (application) layer. The most important candidate on top of these layers is the XML based protocol SOAP (Simple Object Access Protocol) used together with XML application instances. For SOAP a security enhancement is defined, the WS-Security, that supports:

- message integrity
- message confidentiality
- message authentication
- security token passing

SOAP defines a messaging framework expressed in XML and the instances could be validated using an XML Schema (but this is not required). SOAP specifies binding to the HTTP protocol.

The following elements are defined:

- *envelope* – the top element (mandatory)
- *header* – contains control information (optional)
- *body* – contains data (mandatory)
- *fault* – contains fault information (placed in *body*, optional)

Header and body could contain elements from other namespaces than SOAP's own (thus SOAP is really a framework). One example is to include security information in the header since this is not supported by SOAP. Some other XML protocols are:

- Jabber <http://www.jabber.org/about/overview.html>
- Internet Open Trading Protocol <http://www.oasis-open.org/cover/xmlMessagingIETF.html>
- XMSG <http://www.w3.org/TR/2000/NOTE-xmsg-20001013/>

There could also be a need to use remote procedure calls (RPC). SOAP supports RPC and also XML-RPC is a candidate (see <http://www.xmlrpc.com/>) which is an XML application based on HTTP.

7.4 Databases

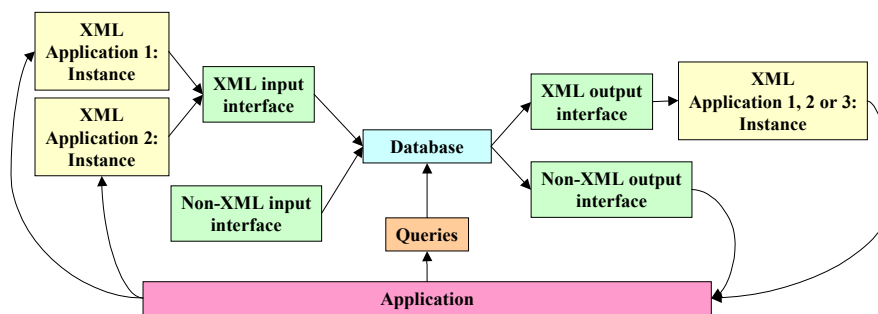
Since a grammar must be created for validation, the normal use of an XML application is to cover many XML application instances. Thus, a need for some kind of database is natural. We

want to store information originally specified in an instance and to generate an instance from information in the database. A practical way to classify the need is to consider the following two general types of databases:

- *data-centric* – here the focus is on data as such i.e. how to store and retrieve data.
- *document-centric* – here the focus is on documents i.e. how to store and retrieve document files / larger pieces of data

We could also have a need anywhere between these two extremes and, further, the need could be varying.

Internally a database could store information according to its own internally specific format or storing information as XML application instances (native XML database). In the former case some kind of middleware is necessary for transforming to/from an external XML world. In the latter case some kind of transformation is necessary for transforming to/from a non-XML world. The picture below shows the principle blocks. The choice of internal representation is in principle a performance issue (in terms of time, memory and other properties). However, one important aspect is if processing instructions and comments (they are not part of the instance character data) are handled correctly in a non-native XML database. If preserving XML information is important then a native XML database should be used.



A natural starting point for queries is the XML Query Language (XQuery) (see [14]) another is the XML extensions made to SQL (denoted SQL/XML). A natural starting point for interfaces needing transformations is XSLT.

For commercial database products a reference to an extensive list is given in [45].

7.5 Web services

One general problem today is to enable communication between different systems in a safe and secure way. Systems can be local or distributed and especially when including the Internet the need grows strong. Since the systems have generally been developed with different purposes, by different companies and using different hardware and software it is necessary to find a standardised way that everybody can agree upon.

We see the need of using functions via the Internet i.e. *web services*.

A web service is a package of functions, within a specific application area, which can be used by other functions via the Internet.

Since XML is a world wide standard a web service XML application has the potential of being the ideal candidates for communicating over the web. Web service input and output data could then be sent according to the specified XML application.

One example of an XML application in this area is WSDL (Web Services Description Language see [7]) which describes web services. The need for such an XML application can be described by:

- A potential client of the web service has to understand the functionality and the properties of the service.
- A potential client of the web service has to know the inputs and outputs of the service i.e. the interface. This includes both syntax and semantics i.e. how to specify parameters and what they actually mean.
- A potential client of the web service has to find the service (actually this is not addressed by WSDL, instead WSDL could be used by an agent searching for services).

By defining a standardised way of transferring such information the client and server will have the same opinion on the interface from both a semantic and a syntactic point of view. The principle is that the web service server provides a specific instance (according to the format given by the XML application). The client can read it for understanding the use of the web service.

In order to make it possible to find web services there is the UDDI (Universal Description Discovery Integration) initiative see <http://www.uddi.org>. The intent is to provide a global registry for web services like “yellow pages”.

However, one has to be careful when using external web services in applications:

- Access to the web service must be guaranteed (according to some measure).
- Execution performance may vary due to others using web services on the same external server.
- Security and other qualities must be guaranteed (according to some measure) by the company providing the web service.
- Web service development takes place without control by the users.

Thus one has to thoroughly analyse the consequences of placing a part of the application externally.

7.6 Security

7.6.1 Introduction

Here we consider security aspects on the XML application instance level. We must point out that encryption could be implemented within the protocol stack as well at different levels. Thus there is a choice affected by e.g. encryption execution overhead, system design and security requirements.

By security is here meant how to transfer information in a secure way. The means for handling security issues are:

- Digital signature
- Encryption

- Support for access control, authentication

Generally we have the following aspects:

- *integrity* – how do protect data from being manipulated
- *confidentiality* – how do we keep information secret
- *authentication* – how do we know for sure who has sent the information
- *authorisation* – how much is an authenticated person allowed to do
- *non-repudiation* – how can we verify that information actually has been sent, how can we verify that information actually has not been sent

The question is the relative importance of these e.g. is it enough to detect if changes have occurred during transmission and then require retransmission (i.e. an integrity issue) or is confidentiality just as important?

Orthogonal to the aspects above we have some major concerns for making the handling applicable:

- The granularity i.e. how small parts shall be considered. For example, shall an element and its contents have its own encryption and digital signature? By using fine granularity it is possible to have a single XML application instance that can be circulated between many different parties and where each is responsible just for its own part.
- The order of applying means e.g. shall encryption always be handled last? Shall functional contents always be handled first?
- How high is the performance overhead? Shall only the key be transferred using best possible encryption, i.e. with high overhead, and the key is then used for a decryption algorithm with low overhead?

Since these questions might be strongly connected to the use of the XML application separate guidelines might be needed that are outside the definition of the XML application.

To exemplify security aspects we can describe a number of scenarios that are applicable when using public key cryptography. This algorithm is described by:

- Two keys are involved; a private key and a public key. They are associated with each other. Even if knowing one of the keys it is very difficult/impossible to find out the other.
- Data flows from those having the public key to the holder of the private key.
- The holder of the private key keeps the private key secret.
- Any sender, sending data to the receiver, uses the (same) public key for encryption.

Sending data:

1. The sender sends encrypted data using the public key.
2. The receiver decrypts the data using the private key.

Sending signature:

1. The holder of the private key sends an encrypted signature using the private key.
2. The receiver of the signature decrypts the signature using the public key.

If the received signature is valid it must have been issued by the holder of the private key.

Non-repudiation: Receiver claiming (falsely) that data was not sent

1. The sender sends encrypted data using the public key.
2. The holder of the private key sends a signature as an acknowledgement of received information.

If the received signature is valid it must have been issued by the holder of the private key, and here, as an acknowledgement.

Non-repudiation: Sender claiming (falsely) that data was not sent

1. A new pair of keys are used with the new private key at the sender and the new public key at the receiver.
2. The sender sends encrypted data using the original public key together with a signature encrypted using the new private key.
3. The receiver decrypts the data using the original private key and decrypting the signature using the new public key.

If the received signature is valid it must have been issued by the holder of the new private key i.e. the sender.

For the security within a system, security aspects related to XML is just one part of the total security. System design, operation and maintenance are other important aspects which can compensate and/or complement XML related security properties.

7.6.2 Encryption

Here we consider encryption at the XML level and specifically the W3C recommendation XML Encryption Syntax and Processing (see [34]). Encryption can take place for:

- The whole document
- The element i.e. including the element name
- The element content i.e. leaving the element name clear but not the content.
- Other types of data e.g. sequence of characters, binary data.

The sequence for encryption is described by:

1. Select the item to be encrypted and perform the encryption outside the XML application instance.
2. Replace the item with the encrypted version and add encryption information in order to enable the receiver to decrypt it.

The sequence for decryption is described by:

1. Use included encryption information for decryption.
2. Replace the encrypted version with the item in clear text.

Note that, at encryption, when all replacements have been made the document is still an XML application instance however with hidden information. Thus applying XML support as described earlier will generally fail e.g. when trying to find elements, relating ID and IDREF etc. There is also another complicating matter. Since there might be different parsers etc at the sender and receiver sides some minor changes might occur that are not significant but makes sent and received versions different. For example, there might be white spaces removed/added. Thus decryption will fail unnecessarily. A solution to this is to transfer the canonical representation (this was described earlier) instead of the original XML application instance.

Chapter 5 in [34] describes a number of encryption algorithms and chapter 6 contains considerations and recommendations.

7.6.3 XML Signature

Here we consider signatures at the XML level and specifically the W3C recommendation XML-Signature Syntax and Processing (see [31]). Signatures can be used for any type of data.

Signatures use XML elements and below is a principle figure shown.

```
<Signature>
  <SignedInfo>
    ...digest value 1 of part 1...
    :
    ...digest value v of part n ...
  </SignedInfo>
  ...signature value...
</Signature>
```

Control information is also needed, e.g. for specifying algorithms, but not shown above. A digest value could be seen as a checksum of the corresponding part. A part is referenced via a URI. In the same way as for encryption the canonical representation is necessary to use for calculation of digest values. Thus there is a two-step process: first calculating digest values for all parts of interest and then to create the signature for all these parts taken together. At verification there are the corresponding two steps: first verify the signature and then verify each digest value.

Chapter 6 in [31] describes a number of algorithms.

7.6.4 Encryption and signature taken together

We see that encryption and signature handling are to a large extent independent of each other and this creates a number of potential problems. Some general aspects are:

- Adding a signature means a guarantee of the content and thus it should be visible i.e. not encrypted (for a human or computer) at the time of signature.
- Nesting is possible both for encryption and signature. For example, a signature could be made for a number of encrypted elements, encryption can be made for a number of signature handled elements and encryption could be made on a mix of already encrypted and signature handled elements.
- The order of encryption and signature could be crucial. For example, if the contents of an element is encrypted it is necessary to know if signature creation takes place before or after encryption.
- For both encryption and signature it is necessary to clarify the need of canonical representation i.e. if there exists more than one version of an XML application instance with the same logical content but not being identical.

7.6.5 Access control

One possibility is to use Security Assertion Markup Language (SAML) see [11]. It contains ca 50 pages. SAML includes request-response support and is typically used for handling web services. SAML is XML based and several elements and attributes are defined.

SAML supports that a client can call an “authority” and get the privileges it is allowed to (a security context) for accessing other resources (thus only a single sign-on is needed). In this way resources can be security related for a specific client.

SAML handles authorisation and authentication but does not consider other security aspects such as encryption and signatures.

Another possibility is to use Extensible Access Control Markup Language (XACML) see [13]. It contains ca 130 pages. XACML is motivated as a language for expressing security policies. Using XACML at all places in a company makes it possible to create a common view on security matters. XACML supports access control down to single elements.

7.7 User interface

XForms (see [46]) is an XML application to be used within other XML applications and especially within XHTML. Xforms corresponds to forms in HTML. Implementations are described in [47].

Note that XForms is a recent recommendation but, as it seems, there is no real competitor.

7.8 Application Programming Interface

7.8.1 Introduction

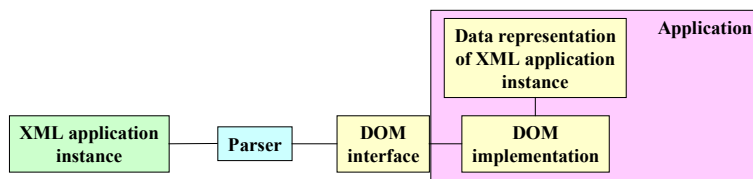
A natural start approach for an XML programming language is Java since it is to a large extent platform independent (although not completely). Thus Java matches the platform independence of XML application instances. It could also be interesting to consider the .NET technology from Microsoft.

Below the DOM and SAX interfaces are presented. As will be seen they represent two opposite views how to handle XML application instances in an application.

Since the actual purpose of using DOM and SAX is to extract and/or modify information an alternative could be to use script languages directly on the XML application instances. Some script languages are specifically designed for texts (e.g. Perl) and thus matches XML directly.

7.8.2 DOM

DOM (Document Object Model, see [51]) is an application programming interface that matches a complete internal data representation of an XML application instance. DOM is handled by W3C and development is reflected by levels where 1, 2 and 3 exists at the moment and where level 1 and 2 are official W3C recommendations. DOM uses objects for the data representation and in the object oriented sense. Thus it is natural to use an object oriented programming language such as Java. An object encapsulates data and provides the methods for handling it. The figure below shows the overall idea of DOM.



The internal data representation makes it possible to have full control of the information. This principle results in the following properties:

- Results of parsing could take a long time before they are available and is directly proportional to the size of the XML application instance.
- The application can modify the structure of the data representation and/or handle the data in any desired way.
- A large memory is needed, the whole representation has to be built up even if only parts of it are of interest.
- Once the data representation is created there is a lot of freedom, DOM supports flexibility.

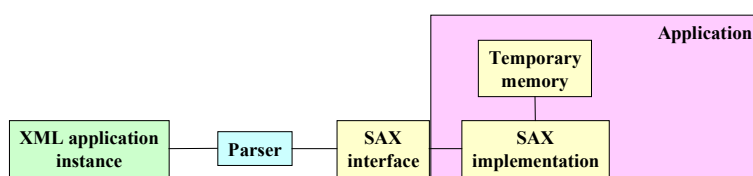
There are browsers, e.g. Microsoft Internet Explorer, that support DOM.

7.8.3 SAX

SAX (Simple API for XML, see [50]) is an application programming interface that uses an event-based interface and intended for the (object-oriented) programming language Java. SAX is easy to use and is not an official standard at least not yet. The main principle is that parsing takes place as a sequence without building up a complete internal data representation. This results in the following properties:

- Results of parsing starts immediately; there is no wait for building up information
- It is easy to discard irrelevant information
- No large memory is needed
- The actions taken as a result of parsing must be thoroughly mapped to the sequential behaviour in order to not have to parse more than once
- SAX is more static and less flexible than building up a complete structure to work on e.g. modifications are not supported

The figure below show the overall idea of SAX.



The parser adapted to SAX sends events, e.g. detection of start/end of element, attributes etc, according to the SAX interface to the SAX implementation. The implementation is a part of the application and is used for making application dependent processing as a result of parsing an XML application instance. One example could be to compile information stored within

several different element contents. Since parsing here is just a sequence from beginning to end of the whole XML application instance it is generally necessary to keep some context information in a temporary memory. For example it could be necessary to remember ancestors to the current element in order to process it correctly (a stack is a good way of representing such tree structure information).

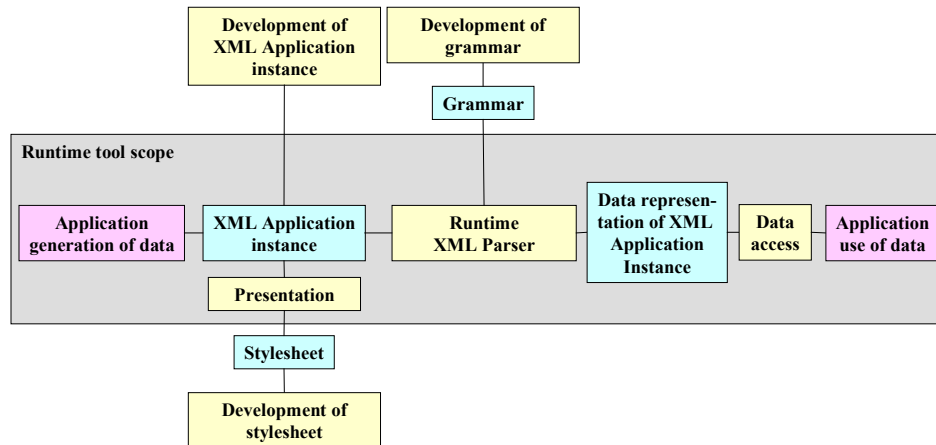
7.9 Conclusions

In this chapter different system and principle implementation aspects have been considered. These reflect an application that is distributed and where security is a major concern. Especially when using Internet in an application these aspects become crucial. By going through the content of this chapter it is possible to define the scope of the application thus making it possible for everyone involved to get a picture of the system. Even if not all aspects are relevant it is anyhow of high value to elicit them and to perform deliberate top-down decisions what shall be included and not.

8 XML tools

8.1 Introduction

The picture below shows the applicability of different tools and we can separate two phases: *development* and *runtime* where the latter is a subset of the former as shown in the figure.



From the figure we see that a parser is used in two ways:

- Explicitly for creating a data representation of the XML application instance. In this case the marked up *data* is of interest and used by the application. This occurs at development and at runtime.
- Implicitly for verifying if the XML application instance is valid or not. In this case the validated instance is used for tests or for other purposes e.g. as support to the runtime XML parser. Two examples are XML Schema and XSLT. We will assume that validation is only relevant during development since the application is assumed to handle this aspect at runtime e.g. by signalling an illegal input to the operator.

Note that protocol aspects, such as how to transfer data using e.g. SOAP, are not included here since they are considered implementation issues.

Below principal functionality will be described. The list is not complete but shows the most important and common functions. In general, commercial tools include several functions especially those classified as Integrated Development Environment (IDE)

8.2 Development of XML Application Instance

The functionality could include:

- Editing
- Validation
- Conversion to canonical form
- Support for Xlink, Xinclude, RDF etc
- Support for XPath, XPointer (non-XML) editing and displaying expressions and their results

- Support for checking of differences, i.e. if a file has been modified, and merging for consistent updates
- Support for digital signatures, encryption, authorisation
- Conversion from non-XML or another type of XML application instance into a new one XML application instance
- Compression tool compressing XML application instances
- Search engine for information in XML application instance

8.3 Development of grammar

The functionality could include:

- Editing (e.g. DTD, XML Schema) possibly including conversions between them (and maybe also other types of grammars)
- Parsing
- Description of grammar for human reading
- Creating grammar from XML application instance
- Conversion from one type of grammar into a new one

8.4 Development of stylesheet

The functionality could include:

- Editing (e.g. CSS, XSL, XSL Formatting Objects, XSLT)
- XSLT processing i.e. transforming XML information to e.g. HTML
- Debugging (e.g. XSL, XSLT)

8.5 Presentation

The functionality could include:

- Using HTML with a browser
- Support for printout
- Support for forms

8.6 Data access (API)

The functionality could include:

- Implementation of e.g. DOM, SAX.

8.7 Tool links

There is a huge amount of XML tools available and the links below is by no means complete nor does it reflect product quality or importance. However, it is assumed that the most common tools are included.

First some links to XML portals are listed containing a lot of XML information and tools:

- <http://xml.startkabel.nl/>
- <http://www.flashtastic.org/html/xmllinks.htm>
- http://www.flashdeveloper.nl/g_XML_portals.html (a link to portals)

- <http://www-106.ibm.com/developerworks/views/xml/tools.jsp> (IBM specific)

A list of free XML tools, maintained by a private person, is found at http://www.garshol.priv.no/download/xmltools/cat_ix.html

XML Software, at <http://www.xmlsoftware.com/>, maintains a list of tools where people could submit information on new tools.

Tools more specifically related to C and C++ can be found in <http://www-106.ibm.com/developerworks/xml/library/x-ctlbx.html>

The following links are also good information sources :

- <http://www.mds.rmit.edu.au/~tja/aija-vscl-2001/index.html>
- <http://xml.uni-muenster.de/tools/>
- <http://www.about.reuters.com/researchandstandards/events/2000/11/bite/seminar.htm>
- <http://www.lib.uwaterloo.ca/~cpgray/xml4lib.html>
- <http://www.perfectxml.com/software.asp>
- <http://www.xml.com> (look at Buyer's guide)
- <http://www.xmlpitstop.com/xmlTools.htm>
- <http://www.xmlphant.com/pages/Tools/>

8.8 Conclusion

From above we could see that there is a huge amount of tools available with varying qualities and prices. However, when looking at those dominating commercially there are not so many. A question is also if free software tools can be used in real, perhaps safety critical, applications. It is recommended not to choose them due to the following potential problems:

- Lack of maintenance
- Lack of updated versions
- No one responsible for the correct behaviour
- No quality assurance.

We can give some questions that could be used for guiding the tool decision:

- How flexible shall the application be?
- What XML support is relevant?
- What “nice features” shall be included? Are they really necessary? Now or in the future?
- Can you define your demands for *development* and *runtime* separately? What are they?
- Are there other considerations apart from functional when choosing tools?
- Shall tools be standalone or shall they be integrated with application software?
- Can you state for each tool: input, output, importance, valid for phase, changes in future use, degree of integration, performance requirements?

Where should you start when choosing tools? One introduction is given in [41] even if not completely up to date.

A second alternative is to study the references above; if a tool appears in many lists it is probably a sign of commercial success and thus could be an initial alternative.

A third alternative is to study books since it is more likely that they included tool descriptions there.

A fourth alternative is to start considering some well-known tools for getting acquainted with capabilities and qualities. An initial approach could be:

- Internet Explorer (<http://www.microsoft.com/windows/ie/default.asp>)
- XML Spy (<http://www.xmlspy.com/>)
- XML Pro (<http://www.vervet.com>)
- XMetal (<http://www.corel.com>)
- Sonic Stylus Studio (<http://www.stylusstudio.com/>)

A fifth alternative is to co consider the work within the Apache XML project (see <http://xml.apache.org/>) since they have several different kinds of XML tools.

9 Creating an XML application

9.1 Introduction

In this chapter we highlight some aspects that should be considered from the following point of views:

- Is the aspect relevant for the XML application or can it be in the future?
- Is the aspect relevant in another form?

9.2 Data model

Before any real work can start on the XML application it is necessary to define a *data model*. As the name suggests it is a model of the data that is underlying the structure of the XML application. The data model spans the total scope of the area of interest and could be expressed in any format e.g. structured English. Apart from being the basis of XML application there are other uses of the data model:

- To make the information available to those not interested in XML syntax
- To compare the current data model with other ones
- To test scenarios for verifying data completeness, uniqueness and applicability
- To define the granularity of information
- To define future extensions
- To define transactions

9.3 Overall design principles

Below is a number of considerations and recommendations given without any priority. Both the XML application as such and the supporting system are included.

One of the most fundamental questions is who is actually to read the XML application instance? Is it designed for humans, computers or both? How often will humans and computers read compared to each other and compared to writes? It is thus necessary to analyse to what extent readability shall be supported. For example, shall namespace prefixes be used explicitly or shall default values be used as much as possible? Is the end-user interested or is it just the developer? The decision will govern the overall design of the XML application.

Which kind of grammar shall be used? Is there a good motivation? See the discussion in the corresponding chapter earlier.

Extensions are natural in the XML world. How shall extensions be supported? How is extension affected by the choice of grammar (normally DTD or XML Schema)? Is it possible to first create a small version of the XML application and extending it in a natural way in the future?

Keep the number of structures down in order to limit maintenance work. A structure occurs as soon as there are dependences between separate items. Some possible structure examples are entities, grammars and fragments of XML application instances used for creating new ones.

Before creating the XML application some kind of underlying data model must be defined. Is there a one-to-one mapping between the data model and the XML application? Why / why not?

How is the balance made concerning security between:

- XML application and its instances
- Protocols
- System design e.g. network issues
- Operator work procedures
- COTS tools and software
- New software/hardware developed within the system

9.4 Use of design patterns

The origin of design patterns stems from Christopher Alexander who defined them in relation to architecture. The idea was brought over to the software community and applied to specific software design constructs that occurred over and over again but in different shapes. One example of a software design pattern is a wrapper that could be used for maintaining a stable interface even though the interior is changing. Design patterns have the potential of allowing future modifications in a controlled manner and to improve structure and reusability. See [38] for discussions concerning patterns for dealing with change.

To describe a design pattern there is a more or less standardised way using the following headlines:

- Name
- Problem
- Context
- Forces
- Solution
- Examples
- Resulting context
- Rationale
- Related patterns
- Known uses

Since a new XML application (and its instances) is also a design it is natural to transfer the idea of design patterns to the XML world. The list below is from <http://xmlpatterns.com/> and shows a representative collection of design patterns together with short descriptions.

Catch-All Element	A container element for dealing with unknown elements within the document.
Choice Reducing Container	When creating large DTDs, authors are required to learn a large number of attributes and elements, and where they can be positioned in order to know how to use the DTD. By reducing the number of choices that the author has to make at any point in the DTD by group related elements beneath newly introduced elements,

	the learning requirements of the author can be reduced.
Collection Element	Create a new element whose content model allows only instances of a single element type.
Common Attributes	Provide a set of attributes that can be placed on all, or most, elements in the document type.
Consistent Element Set	Provide a set of elements which is consistently grouped together as the content models of a number of other elements.
Container Element	A container has multiple elements as child elements. A new element type is created to group related elements. This is a very general pattern and many other patterns specialize this one.
Content Type Label	Parameter entities are created to represent different types of values within a DTD.
Declare Before First Use	Elements which are referenced by another part of a document should be found earlier in the document than the first place they are referenced.
Domain Element	A concept from domain analysis is made into an element.
Envelope	Provide a document type which is defined to be a holder for other, arbitrary XML data.
Extensible Content Model	Provide a mechanism which allows additional elements to be added into existing content models.
Flyweight	If the same information is included at many different points in a document the information can be placed in just one place, and shared from each place in the document that needs to refer to it.
Generic Element	To provide flexibility to users of the document, designers can provide an element type that is very generic. The use of the generic element is not well specified by the documents type. This allows for authors to use the document type in ways that may not have been foreseen.
Head-Body	When a large amount of metadata needs to be included in an element the designer may create two children for the element, one for the metadata and one for the body of the document.
Marketplace	Instead of organizing objects in a hierarchical fashion, objects are organized in a linear way, with signs on each object to indicate its classifications.

Metadata First	Metadata should appear in a document before the data which it is about.
Metadata in Separate Document	When there is a large amount of metadata for a document it can make sense to separate this into a different document altogether, with its own document type.
Multi Root Document Types	A single document type with multiple root elements is used to handle varying documents that may exist within a system.
Multiple Document Types	When a system needs to represent a range of different document types, each document type can be represented by a completely separate declarations.
Optional Container Element	When creating large DTDs with many logical units authors might be required to learn a large number of these logical units to know how to use the DTD. By hiding the details of optional parts of the DTD beneath optional elements, some of this complexity can be reduced.
Parallel Design	Creating structures that for different elements that are very similar to one another makes DTD easier to use and understand.
Referenced Note	When an area in text needs to refer to a note that will be placed in a different part of the document, a reference is made to a separate entity using an IDREF.
Reuse Document Types	If document types already exist for the job at hand, they can be reused completely or parts of them can be reused.
Role Attribute	Sometimes the designer of a DTD can not foresee all of the needs that the author of a document will have. In order to give the author flexibility an attribute that specifies a role can be included on some of the elements.
Separate Metadata and Data	When documents contain content and data about the content, the two types of data should be clearly separated.
Short Understandable Names	Names of elements and attributes should short and understandable by authors and developers of processing software.
Universal Root	Provide a single root element that contains an option of multiple elements. Often used for different transaction types within a single document type.
Use XML	XML technology can be used to represent structured information. This pattern helps determine when XML is an appropriate solution.

9.5 Preparing for extensions

9.5.1 Introduction

An important aspect is if an XML application is extensible by just making additions i.e. if an original XML application instance (file) could be kept intact and additions are used only by new instances (backward compatibility). If modifications are necessary, the changes must be clear and if not made correctly an error must be generated.

There are many aspects that affect the choice where to place extensions and two important aspects are to limit complexity and dependences. Principles for extensions must also consider implementation issues. These include:

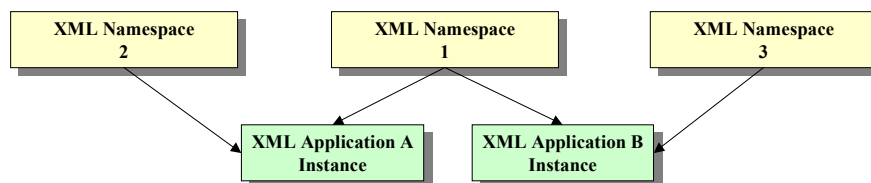
- coding
- verification and validation
- quality
- compatibility

9.5.2 XML

Generally, extension possibilities depend on the chosen grammar and XML as such. From extension point of view XML Schema is superior compared to DTD. For example, the possibility of complex types in XML Schema and the use of inheritance. For extending the XML application as such we can list some aspects:

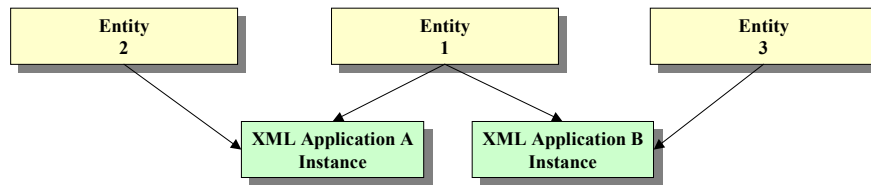
- To add more processing instructions and notations.
- To allow new elements
- To allow new attributes names/values e.g. enumeration values
- To give new choice alternatives for element content
- To change occurrence e.g. from zero-or-one to zero-or-more

There are also more external aspects to consider. Including new namespaces is probably the most natural way to enable extensions. An example is given in the figure below.



We see that it is possible to create structures with complex dependences e.g. two different XML applications can become dependent. Stub namespaces could be defined for future use in order to flag for probable extensions.

External entities could also be used for extensions in the same way as namespaces. Changing the content of an entity makes it possible to extend functionality.



By having external access we get the following properties:

- It is easier to create some kind of standard since others could use the same item.
- Changes in the common item will affect all users thus the item must be very stable.
- By using a common item, constituents could be overridden locally i.e. a form of inheritance.
- The server containing the common item must be stable in order to always be available.
- Spread of virus, advertisements etc could be an issue.
- We have to handle versions of separate items correctly.

If backward compatibility is not an issue we could instead add restrictions in future versions and the main reasons for this could be:

- More knowledge is gained concerning the use of the XML application
- Too much freedom was allowed creating compatibility problems
- The unrestricted item was there only to flag that “something has to be done for this in the future”

Some examples are:

- Replace default values with individual values
- Replace element content Any, Mixed, Empty with more accurate descriptions
- Remove external entities and other extension possibilities that were not really necessary

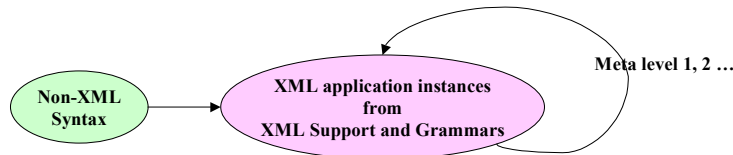
9.5.3 XML Support

Here we consider extension possibilities that are outside the actual XML application instance. Extensions here means possibilities to extend the use of the instance as such.

Transformations could be defined that makes it possible to use the current instance by others and/or for the current instance to use others. The reasons could be:

- To import/export data to/from other new applications
- To increase the number of presentation alternatives

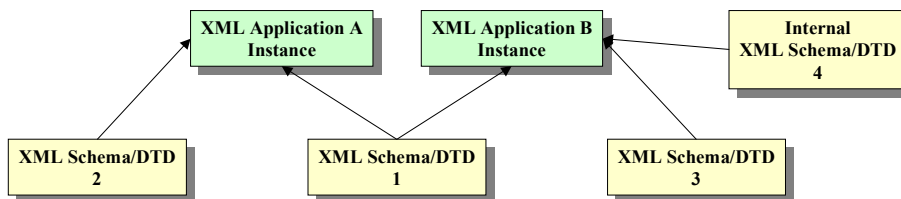
From the previous discussions we see that several of the support functions of XML were expressed as XML applications instances. Thus it is possible to create meta XML support. One example is that we could use XInclude to merge different XML Schemas into a new one e.g. an overall XML Schema template, an XML Schema library etc. A bit more fare fetched is to continue this principle with meta meta XML support and so on, however, these possibilities are also included here. The picture below shows the principle.



Another example, as mentioned above, is Schematron which uses XSLT for first creating a “meta stylesheet” which is then used for creating the actual stylesheet used for validation. Thus extensions are here made by increasing the abstraction level or by creating new compositions.

9.5.4 Grammars

More than one grammar could be used, own developed or external standard, perhaps including a reference to a dummy grammar that will be filled in the future. This makes it possible to implement override and inheritance. Note that external parameter entities also could contain declarations i.e. parts of grammars. An example is given in the figure below. We see that it is possible to create structures with complex dependences.



9.5.5 System design and implementation

Extra functionality could be implemented e.g.

- as code in CDATA sections e.g. using script languages
- as RPC
- using information in Notation specifying a URI for the executable
- using processing instructions specifying a URI for the executable

9.6 Documenting an XML application

There are a lot of standards and recommendations to consider. Many of them are connected in a “spaghetti-like” relationship. A probable motivation for reading this report is to create another recommendation, official or not. Since the ideas behind a recommendation are not difficult but when including all details the recommendation gets much more complicated there are some points to remember:

- Examples *are* the silver bullet for a reader to understand what it is all about. Include comments when necessary.
- Try to build up successively i.e. minimize forward references (the number of forward references is probably a good metric for readability). Also let main ideas come first without discussing exceptions.
- For an HTML-version on the web use links so it easy to find definitions.
- Use pictures for describing dependences.

Or, as a summary, if you can fully live up to the following principle you will get a success:

It shall not be necessary to understand the document before reading it!

9.7 Checklist before making things too complex

As we have seen it is very easy to get started e.g. by creating your own XML application in a limited way. Including the full potential requires much more effort but there is a good chance that even the limited way will fulfil or nearly fulfil all your needs. However, looking at the future development might complicate matters. One significant complexity measure is the number and size of structures that have to be maintained due to the design. Some example structures are:

- Attribute sets and the nesting of them
- Inclusion mechanisms e.g. XInclude
- External entities
- External resources and relations between them
- Design patterns
- Inheritance hierarchies
- Nesting rules for elements
- Use of default values e.g. for namespace prefixes
- Structure of related XML application instances, grammars etc

The list below shows a number of questions to ask before decision:

- Is the new feature *really* needed or is it just nice to have? Test by presenting your motivation to a colleague!
- What are the consequences when including the new feature concerning: understandability, readability, complexity, consensus, compatibility, performance etc?
- What structures are affected by the new feature?
- Is there a simpler way (even if less beautiful) that does not increase complexity but has some other less important disadvantages such as being more verbose?

See also [37] for discussions concerning avoiding complexity.

9.8 Making the XML application a standard

One obvious organisation for establishing a standard is W3C. However, W3C has its main interest in XML core technology e.g. XML as such, security standards, XPointer etc. There have been some attempts for publishing XML application documents from members in the form of Notes. This will change and W3C will only consider XML core technology in the future.

The natural organisation for an application oriented XML application is instead OASIS. The work of creating a standard is organised within a Technical Committee (TC) for which a process is defined (see [49]) specifying the organisation and structure of work. The standardisation process consists of two steps:

1. Establishing a Committee Specification
2. Establishing the Committee Specification as an OASIS standard

An OASIS discussion group could be started before the TC in order to find out those interested in the XML application. In [48] guidelines are given supporting the TC process.

The OASIS idea is to first create a TC that will then develop a standard but there are several cases where a preliminary standard proposal existed before the TC was created. Membership in OASIS (open to everyone) is necessary and a fee has to be paid every year. Templates exist and instructions for using them can be found at <http://www.oasis-open.org/spectools/>. Membership information can be found at <http://www.oasis-open.org/join/>.

OASIS has a registered namespace where a specific branch can be defined for a new XML application.

Note that OASIS encourages information openness and democracy.

9.9 Conclusions

In this chapter guidelines have been given concerning the creation of a new XML application i.e. a new language based on XML. Especially important aspects are the future use of the XML application and the extension capabilities. One way of testing these is to create likely *extension scenarios* and see how they could be mapped in the proposed structure. If these mappings can be made without increasing complexity the proposed XML application is probably mature. Another, just as important aspect, is to analyse the created structures. Are they too complex today and will they be difficult to maintain? If this is true a restructuring should be made as early as possible since this will be an extremely cost-effective solution.

10 Conclusions

The things that started so simple turned out to be a large world of recommendations and complex relationships, how could this happen? Is the XML too complex and something simpler will come? Or is the complexity an illusion, just an effect of trying to be complete here even if not needed for any normal application? Below we will try to answer these questions and come up with some conclusions.

First of all we must emphasize that it is really easy to get started using XML. This makes it possible to have a constructive dialogue right from the beginning at development. This corresponds to the idea of “rapid prototyping” e.g. in software development in general. Since XML application instances contain plain text everybody could read and understand the purpose and contents of files, thus improving quality. On the other hand, the risk is high that you create a *de facto* standard that is accepted by everyone too early not considering all aspects since they were not known or thought of. A possible solution to this is to plan for extensions right from the beginning by creating stubs, leaving things explicitly open etc. If this is possible everybody involved will note that things might change later on during development and get accustomed to it.

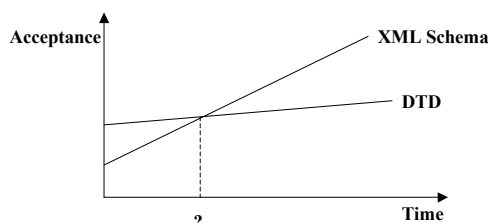
What are the origins to the complexity and what will people feel is too complex? We can give a list of some relevant aspects:

- It is difficult to find a starting point for studies and proceed in a sequential manner because everything is related.
- It is difficult to see what is really needed and what is just nice to have.
- People use different names for the same thing e.g. root element / document entity, / document element.

We have also seen that there is a lot of XML applications and more are produced continuously. Some readers of this report will create other ones. There are two important aspects that should be kept in mind when creating a new XML application:

- Is there an already accepted standard that is possible to use wholly or partially? If not possible, is it possible to give an accurate motivation?
- How can a new XML application be set as a standard in order to allow others to use it? Should the new XML application be modified or adapted in some way in order to improve the chance of being a standard?

In any case the grammar must be specified for validating the XML application instances. Currently the two realistic concepts are DTD and XML Schema. The figure below shows the principle relation between them.



XML Schema will win in the long run but both will remain on the market e.g. tools will support both a long time from now. Further, many tools can transfer more or less

automatically between the two making the decision less critical. However, if extensibility and flexibility are of strong importance XML Schema is directly recommended. A subset could then be used at the beginning.

When including an XML application during development of a system there are many aspects to consider. Just for the parser we can list a number of important ones:

- Does the parser support the needed namespaces?
- For relatively new recommendations, will the parser support all functionality? How does the parser handle changes in recommendations?
- What kind of parsing is preferred? The DOM or the SAX style?
- How big are the files to be parsed today, tomorrow?
- Will different parts of the system use different kinds of parsers?
- Are there parts of the application that cannot be controlled e.g. the use of an external web service outside the project.

and there are corresponding questions for other types of tools.

How can this report guide? The most beneficial approach is to treat this report as an extended checklist which includes motivations and considerations. The intention is to cover all relevant aspects top-down, but not including any details, and hopefully there are no missing aspects. If this is fulfilled the report can be surveyed and each aspect could be marked relevant or irrelevant for the current application thus giving a considerable help before development actually takes place. Making decisions top-down saves money, time and effort!

How is this report of value for work within a project like MANATEE? First of all, the information given in this report is the basis for all discussions concerning the use of XML within MANATEE. Important to all participants is to have a common understanding of the subject as such and how different items can be used in an optimal and top-down manner. Second, looking at possible extension possibilities gives important feedback to system design and compatibility with already existing systems. For example, how shall transformations using XSLT be handled? Third, top-level implementation issues have to be decided early e.g. the principles for data bases. This report will guide and make considerations explicit. As a summary, it is of crucial interest that each participant reads this report in order to simplify and improve discussion and decision quality.

11 References

- [1] Using XML
Second edition, 2002
David Gulbransen et al
ISBN 0-7897-2748-x
- [2] From The Cover Pages
<http://www.oasis-open.org/cover/xml.html#applications>
- [3] The Cover Pages
<http://xml.coverpages.org/>
- [4] Machine Control via Internet – A holistic approach
Lars Strandén, Johan Hedberg, Håkan Sivencrona
SP Report 2002:30
ISBN 91-7848-921-0
- [5] The “Pros and Cons” of XML
ZapThink 2001
<http://www.zapthink.com/reports/ProsConsXML.pdf>
- [6] Extensible Markup Language (XML) 1.0 (Second Edition)
<http://www.w3.org/TR/2000/REC-xml-20001006>
- [7] Web Services Description Language (WSDL) Version 1.2
W3C, Working Draft 3 March 2003
<http://www.w3.org/TR/2003/WD-wsd112-20030303/>
- [8] Voice Extensible Markup Language (VoiceXML) Version 2.0
W3C Working Draft, 23 October 2001
<http://www.w3.org/TR/2001/WD-voicexml20-20011023/>
- [9] Synchronized Multimedia Integration Language (SMIL 2.0)
W3C Recommendation 07 August 2001
<http://www.w3.org/TR/2001/REC-smil20-20010807/>
- [10] XHTML™ 1.0 The Extensible HyperText Markup
W3C Recommendation 26 January 2000, revised 1 August 2002
<http://www.w3.org/TR/xhtml1/>
- [11] Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)
OASIS Standard, 5 November 2002
<http://www.oasis-open.org/committees/security/docs/oasis-sstc-saml-core-1.0.pdf>
- [12] Web Services Security (WS-Security) version 1.0
April 5, 2002
<http://www.verisign.com/wss/wss.pdf>
- [13] eXtensible Access Control Markup Language (XACML) version 1.0

- 18 February, 2003
<http://www.oasis-open.org/committees/xacml/repository/oasis-xacml-1.0.doc>
- [14] XQuery 1.0: An XML Query Language W3C Working Draft
15 November 2002
<http://www.w3.org/TR/xquery/>
- [15] Namespaces in XML 1.0
World Wide Web Consortium 14-January-1999
<http://www.w3.org/TR/REC-xml-names/>
Namespaces in XML 1.1
W3C Working Draft 03 April 2002
<http://www.w3.org/TR/2002/WD-xml-names11-20020403/>
- [16] XML Namespaces FAQ
Ronald Bourret
<http://www.rpbourret.com/xml/NamespacesFAQ.htm>
- [17] Namespace Myths Exploded
Ronald Bourret
<http://www.xml.com/pub/a/2000/03/08/namespaces/index.html>
- [18] Plan to use XML namespaces
David Marston
IBM Research
<http://www-106.ibm.com/developerworks/library/x-nmspace.html> (Part 1)
<http://www-106.ibm.com/developerworks/library/x-nmspace2.html> (Part 2)
- [19] XML Base
W3C Recommendation 27 June 2001
<http://www.w3.org/TR/2001/REC-xmlbase-20010627/>
- [20] XML Linking Language (XLink) Version 1.0
W3C Recommendation 27 June 2001
<http://www.w3.org/TR/2001/REC-xlink-20010627/>
- [21] W3C Technical Reports and Publications
<http://www.w3.org/TR/>
- [22] XML Pointer Language (XPointer)
Currently four parts exist:
<http://www.w3.org/TR/xptr-framework/>
<http://www.w3.org/TR/xptr-element/>
<http://www.w3.org/TR/xptr-xmlns/>
<http://www.w3.org/TR/xptr-xpointer/>
- [23] The "xml:" Namespace
<http://www.w3.org/XML/1998/namespace>
- [24] XSL Transformations (XSLT) Version 1.0

- W3C Recommendation 16 November 1999
<http://www.w3.org/TR/1999/REC-xslt-19991116>
- [25] XML Fragment Interchange
W3C Candidate Recommendation 12 February 2001
<http://www.w3.org/TR/xml-fragment>
- [26] Canonical XML Version 1.0
W3C Candidate Recommendation 26 October 2000
<http://www.w3.org/TR/2000/CR-xml-c14n-20001026>
- [27] The Annotated XML Specification
Tim Bray
<http://www.xml.com/axml/testaxml.htm>
- [28] XML Information Set
W3C Recommendation 24 October 2001
<http://www.w3.org/TR/xml-infoset/>
- [29] Cascading Style Sheets, level 1
W3C Recommendation 17 Dec 1996, revised 11 Jan 1999
<http://www.w3.org/TR/REC-CSS1>
- [30] Scalable Vector Graphics (SVG) 1.1 Specification
W3C Recommendation *14 January 2003*
<http://www.w3.org/TR/SVG11/>
- [31] XML-Signature Syntax and Processing
W3C Recommendation 12 February 2002
<http://www.w3.org/TR/xmlsig-core/>
- [32] Extensible Stylesheet Language (XSL) Version 1.0
W3C Recommendation 15 October 2001
<http://www.w3.org/TR/2001/REC-xsl-20011015/xslspec.html>
- [33] Resource Description Framework
<http://www.w3.org/RDF/>
- [34] XML Encryption Syntax and Processing
W3C Recommendation 10 December 2002
<http://www.w3.org/TR/xmlenc-core/>
- [35] XML Schema Part 1: Structures
W3C Recommendation 2 May 2001
<http://www.w3.org/TR/xmlschema-1/>
XML Schema Part 2: Datatypes
W3C Recommendation 02 May 2001
<http://www.w3.org/TR/xmlschema-2/>
- [36] XML Schema Part 0: Primer

- W3C Recommendation, 2 May 2001
<http://www.w3.org/TR/xmlschema-0/>
- [37] W3C XML Schema Design Patterns: Avoiding Complexity
Dare Obasanjo, Microsoft Corporation
<http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlschemacomplex.asp>
- [38] W3C XML Schema Design Patterns: Dealing With Change
Dare Obasanjo, Microsoft Corporation
<http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnxml/html/xmlschemachange.asp>
- [39] Schematron: validating XML using XSLT
Leigh Dodds April 2001
http://www.ldodds.com/papers/schematron_xsltuk.html
- [40] World Wide Web Consortium
<http://www.w3c.org>
- [41] XML Publishing 2002: state of the TOOLS
EMedia Magazine, Robert J. Boeri
http://www.theworld.com/~bboeri/pdfs/emedialia/ii0112-buyer_guide.pdf
- [42] XML Entities and their applications
IRT.org
Pankaj Kamthan, 21st May 2000
<http://tech.irt.org/articles/js212/index.htm>
- [43] Understanding SOAP
Aaron Skonnard, March 2003
<http://msdn.microsoft.com/webservices/default.aspx?pull=/library/en-us/dnsoap/html/understandsoap.asp>
- [44] SOAP Version 1.2 Part 0: Primer
W3C Candidate Recommendation 19 December 2002
<http://www.w3.org/TR/2002/CR-soap12-part0-20021219/>
- [45] XML and Databases
Ronald Bourret, January, 2003
<http://www.rpbouret.com/xml/XMLAndDatabases.htm>
- [46] XForms 1.0
W3C Candidate Recommendation 12 November 2002
<http://www.w3.org/TR/xforms/>
- [47] XForms – The Next Generation of Web Forms
<http://www.w3.org/MarkUp/Forms/>
- [48] OASIS Technical Committee Guidelines

12 Feb 2003

http://www.oasis-open.org/committees/committee_guidelines.pdf

[49] OASIS TECHNICAL COMMITTEE PROCESS

16 September 2002

http://www.oasis-open.org/committees/committee_process.pdf

[50] Official website for SAX

<http://www.saxproject.org/>

[51] W3C home page for DOM

<http://www.w3.org/DOM/>