XML as a Format for Representation and Manipulation of Data from Radar Communications

(HS-IDA-MD-01-301)

Anders Alfredsson
Department of Computer Science
Högskolan i Skövde, PO Box 408
SE-54128 Skövde, SWEDEN

Final year project
on the study programme in computer science 2001
Supervisor: Henrik Engström
Company Supervisor: Thomas Milton

XML as a Format for Representation and Manipulation of Data from Radar Communications

HS-IDA-MD-01-301

Submitted by Anders Alfredsson to the University of Skövde as a dissertation towards the degree of M.Sc. by examination and dissertation in the department of Computer Science.

October 2001

I certify that all material in this dissertation that is not my own work has been identified and that no material is included for which a degree has already been conferred upon me.

.....

Anders Alfredsson

Acknowledgements

I would like to thank my supervisor Henrik Engström for all his good opinions and advice throughout this project. I thank Magnus Thor Helgasson and Mikael Ågren for the illuminating discussions. I would also like to thank my fiancée Joy, for her unlimited patience and support.

Most of all I would like to thank my grandfather Allan who, despite his illness, has given me the strength and confidence needed to finish my work. You are always on my mind.

Abstract

XML was designed to be a new standard for marking up data on the web. However, as a result of its extensible and flexible properties, XML is now being used more and more for other purposes than was originally intended. Today XML is prompting an approach more focused on data exchange, between different applications inside companies or even between cooperating businesses.

Businesses are showing interest in using XML as an integral part of their work. Ericsson Microwave Systems (EMW) is a company that sees XML as a conceivable solution to problems in the work with radar communications. An approach towards a solution based on a relational database system has earlier been analysed.

In this project we present an investigation of the work at EMW, and identification and documentation of the problems in the radar communication work. Also, the requirements and expectations that EMW has on XML are presented. Moreover, an analysis has been made to decide to what extent XML could be used to solve the problems of EMW. The analysis was conducted by elucidating the problems and possibilities of XML compared to the previous approach for solving the problems at EMW, which was based on using a relational database management system.

The analysis shows that XML has good features for representing hierarchically structured data, as in the EMW case. It is also shown that XML is good for data integration purposes. Furthermore, the analysis shows that XML, due to its self-describing and weak typing nature, is inappropriate to use in the data semantics and integrity problem context of EMW. However, it also shows that the new XML Schema standard could be used as a complement to the core XML standard, to partially solve the semantics problems.

Keywords: XML, data representation, data transformation, data semantics, query language

Table of Contents

List of Figures

List of Tables

1 Introduction

The *eXtensible Markup Language*, hereafter referred to as XML, is constantly gaining more popularity as a markup language for the World Wide Web. Due to its powerful capabilities, XML has become much more than just a temporary buzzword. The concept of XML and all that it involves is now widely accepted not only in the web community, but also in other areas.

Until today the *HyperText Markup Language* (HTML) has been the indisputable universal language for publishing data on the web. However, as applications are beginning to place higher demands on the structure and content of web pages, the simple syntax and functionality of HTML will not be satisfactory (Seligman and Rosenthal, 2001). While the simplicity of HTML has been considered to be its strength and the cause of its widespread use, it is also becoming its pitfall as a markup language for the web in the future.

Standard HTML still has a strong position as a markup language. However, XML with its extensible and flexible structure was built to overcome the shortages of HTML, and is therefore much more suited for the requirements from the applications of tomorrow (Seligman and Rosenthal, 2001). Thus, even if HTML will continue to be the superior standard for the web in the nearest future, XML is predicted to eventually replace it as *the* markup language of the web (Tian et al., 2001).

Although XML was originally created to enable more complex markup possibilities, the potentials of its flexible functionality have begun to breed new expectations, even in the business area (Rosenthal et al., 1999). One of the most common issues is to use XML to straightforwardly exchange data between businesses and thereby develop genuinely inter-operable applications (Abiteboul, 1999).

Since XML offers the opportunity to make a clean distinction between the structure and presentation of data, it brings the data closer to databases (Abiteboul, 1999). This gives the possibility of manipulating the content of the data, facilitating more sophisticated search possibilities than today's web search engines. Another issue, emphasised by Widom (1999), is a speculation of using XML as a format for storing data in different data sources.

The danger of putting too much faith in XML is that it will be misinterpreted as an ultimate solution to the problems of data management and distribution in a company. Surely, the benefits of using XML seem promising, but it will hardly solve everything (Watson, 2000).

A department of *Ericsson Microwave Systems AB* (EMW), a part of Ericsson, resided in Skövde, considers XML to be a potential candidate for use in the work with radar communications. The employees at EMW want to evaluate to what extent XML could be used to solve problems that they have with representation and manipulation of their internal data sources. As a part of this project we will therefore focus on the identification of EMW's requirements and expectations on XML.

The problem considered in this project is to establish the requirements of EMW and to analyse XML to identify the possibilities for representing data in XML format and for manipulation of XML based data sources, in the context of the problems and requirements at EMW.

1.1 Report outline

The remainder of this report will be organised as follows. In section 2 a short presentation of EMW as a company and an insight into the work performed there is given. Section 3 presents the background to XML and related concepts necessary for the understanding of the rest of the report. Section 4 describes general problems in data transformation processes.

Section 5 describes the problem for this project and its motivation, the problem statement and the objectives of the work. The section also presents the methods used for the

objectives. In section 6 a summary is given of the discussions made with key staff members at EMW to get an understanding for their current working situation, and to investigate which their needs are. Section 7 contains an analysis of the extent to which XML can be used by EMW as a means for solving the current problematic situation. The last section, section 8, concludes the work and gives suggestions for future work.

2 Ericsson Microwave Systems

This section gives an introduction to EMW as a company, and to the work performed there. It also presents problems in the work at EMW, followed by a discussion about data transformation in general. Further, a description of a previously analysed relational database solution to the EMW problems is made, along with a discussion about the problems that have been considered to exist for that solution. The section ends with a presentation of EMW's view of XML and its functionality.

2.1 Company introduction

Ericsson Microwave Systems AB (EMW) is a part of the Ericsson company. EMW has its main office in Mölndal, near Gothenburg, Sweden. It also has a small department resided in Skövde. This department will be in focus for the rest of this report. Therefore, the EMW concept will hereafter refer to the Skövde department.

2.2 Work at EMW

The work at EMW consists of project based engineering assignments. The staff is working in small teams with assignments given by clients. One part of this work is concerned with test scenarios for simulated flights and flying tests. The staff in the department that is working with this is concerned with recording the communication between internal radar components in the simulation scenarios. Since the simulated communication takes place on internal network buses, the recording is done by monitoring the network traffic, collecting data sent between components and streaming the results down into big log-files.

A second part of this work is to analyse what happened during the communication scenarios. To be able to do this it is important to get access to the atomic values

characterising the results of the communication. These values are stored in logs and need to be filtered out. The filtration is done by using analysis tools. From the tools it is possible to get the recorded information to be used for different purposes. Examples of such purposes are: optimisation of the simulation, internal verification, and validation against the client or possibly error detection.

2.3 Problems in the work

As the situation is today, EMW has some problems with the way the work is conducted. The logged data has irregular structure and the search and filtering mechanisms in the company are currently not sophisticated enough to fully handle this. There is therefore a substantial need for a new way of filtering out relevant information.

EMW wants to transform data from radar communications into a new format with better structure, which enables more sophisticated searching and filtration on data than is currently possible. This filtration should be done through some sort of query interface.

2.4 An approach towards a potential solution

As an attempt to solve the problems described above, the EMW management formed a small working group and started a project. The aim of the project was to outline the characteristics of a system based on relational database technology, which would be used as a possible solution to the presented problem.

The project group began with analysing and outlining how a relational based system could be designed to be used in the problem context. It was decided that the system should be built from the rules of the relational model and the syntax of SQL92. The group members were themselves going to create a mechanism for transforming the internal data sources into relations. Also, relationships in and constraints on the data should be defined to preserve the internal correspondences of different parts of the data, which was considered important by EMW. The group had limited experience of commercial relational database management systems. Therefore, it was argued that the inclusion of such techniques was to

be kept to a minimum. A system based on basic relational technology with the use of SQL92 was decided to be the most reasonable approach.

The expectations from EMW on the solution were that it would make the internal data sources more perspicuous and better structured than before. Also, since a database system would be utilised, EMW expected that the solution could be used to store large amounts of data more efficiently than was possible with the internal file systems. Moreover, by getting access to the facilities of the SQL92 query language, the EMW personnel believed that more powerful and specific filtering queries could be executed on the data.

2.4.1 Problems with the analysed solution

Having analysed the characteristics of the approach, the project group concluded that a relational database solution would make the internal data very easy and efficient to store and handle, at least to some extent. The SQL92 query language was considered to be sufficient to use for the filtering purposes of EMW.

Although the approach seemed to have many promising features, the group identified some disappointing shortages with it during the analysis. That disappointment initialised a discussion concerning an approach to analyse some other solution for the problems. However, this discussion included no decision to totally abandon the approach. EMW had put too much effort into it to just give it up. Instead, the aim of the project group was to get to know more about strengths and weaknesses of some other possible solution, to avoid a rash settlement for a specific solution.

While analysing suitable ways to model the EMW data as relations, the project group established that a more complex mapping mechanism had to be built than was originally assumed. This was argued to depend on differences between the representation of the internal data structures and the way data would be modelled in the new system.

As a consequence to the modelling differences, the crew encountered problems when trying to represent the more complex kinds of data with SQL92, which had been decided

by EMW to be used for the purpose. They discovered that certain constraining features in SQL92, e.g. structural homogeneity and atomicity of data in tables, made it very complicated to deal with variations and irregularities in the EMW data. Moreover, it was considered hard to define the hierarchical structure of the EMW data with the more flat representation of data in the new relational based solution.

The group also found the relationship modelling possibilities in the new representation to be too narrow. Internal relationships in different parts of the data were not considered to be possible to specify in any satisfying way. Further, the crewmembers wanted to be able to describe properties in the data at a higher level, i.e. to define meta data, to secure the traceability of the data origin. However, they did not see any suitable way to create such definitions in the new system.

The project group agreed that the modelling problems of the new approach could probably be overcome. However, they also argued that this could lead to the creation of unnecessary additional tables and a considerable amount of fields occasionally containing null values. In the long run, this could enforce an undesirable increase in the space cost of a future database system.

2.5 EMW and XML

Due to the problems described above, the EMW management was concerned to analyse some other way to solve the problems in the work at the company. Having browsed the characteristics of XML, EMW grew to have the opinion of XML as being a conceivable solution to the problems. It was therefore decided that it should be evaluated how XML could be used in the same context as was intended with the solution in the previous project. The focus of interest was to get an extensive understanding for XML in the problem context. This specific desire of EMW set the foundation for this project and will therefore characterise the rest of the report.

The status in the area of XML at EMW today is that nobody has ever really used XML in his or her active work. The employees are therefore in need of an introduction to XML and its technologies.

3 eXtensible Markup Language

XML is a standard markup language for representation and exchange of data on the web (McHugh & Widom, 1999). A new Working Group at the *World Wide Web Consortium* (W3C) was formed in 1996 as a response to the new needs of the web. The aim of the work was to develop a new markup standard better suited than HTML for tomorrow's World Wide Web. The work resulted in the XML standard recommendation, which was first proposed in 1998 (Bray et al, 2000).

W3C was also responsible for the creation of HTML (Ragget et al., 1998). The syntax of XML is very closely related to that of HTML, making it easy for users to learn and understand. The big difference is that XML is stricter than HTML, i.e. XML puts more restrictions on the design of documents than HTML does. That and the awareness of the advantages of XML, contribute to greater amounts of data being encoded in XML format.

XML is a subset, or an application profile, of the *Standard Generalized Markup Language*, abbreviated SGML (ISO, 1986). The idea is that every XML document should also be a conforming SGML document. SGML is the international standard for defining descriptions of the structure and content of different kinds of electronic documents (Wüthrich, 1998). It is a system for defining markup languages, and every language that is defined in SGML is called an application of SGML.

Bray et al. (2000, p 4) describe the ten design goals for XML. These are:

1. XML shall be straightforwardly usable over the Internet.

2. XML shall support a wide variety of applications.

3. XML shall be compatible with SGML.

4. It shall be easy to write programs which process XML documents.

5. The number of optional features in XML is to be kept to the absolute minimum, ideally zero.

6. XML documents should be human-legible and reasonably clear.

7. The XML design should be prepared quickly.

8. The design of XML shall be formal and concise.

9. XML documents shall be easy to create.

10. Terseness in XML markup is of minimal importance.

As can be seen, XML has been designed for ease of implementation and for inter-operability with both HTML and SGML (Bray et al, 2000). XML being defined as an application profile of SGML implies that any fully conformant SGML system will be able to read XML documents.

3.1 The HTML Dilemma

To fully grasp the nature of XML it is important to have a complete understanding for why it was originally developed. As mentioned earlier, XML was developed by W3C to prepare for the demands of the future World Wide Web. The motivation for creating XML was based on something called “the HTML Dilemma” (Seligman and Rosenthal, 2001).

HTML is described as the lingua franca for publishing hypertext on the web. It is a very simple markup language focused on handling the presentation and display of data on the web. At the time HTML was created its markup simplicity seemed like a reasonable goal to aim for, and judging from its widespread use that assumption was correct, for the time being (Jansz, 1998). Due to this simplicity HTML allows users to leave the language specification out of the document, and makes it much easier to build applications that process HTML. However, as more sophisticated applications have evolved and the

requirements of using the web are ever increasing, HTML has become deficient in a number of areas. Bosak (1997) discusses three of the most commonly mentioned shortages of HTML. These are:

- Extensibility – HTML is built on a small number of static markup tags for the user to utilise. There is no support for defining new, application specific tags to suit data being represented.

- Structure – HTML does not allow the specification of deep structures, which would be needed to be able to represent database schemas or object-oriented hierarchies.

- Validation – HTML does not allow applications to check the entered or imported data for structural validity.

Seligman and Rosenthal (2001) present another closely related feature that HTML lacks. In HTML the markup is presentation oriented, meaning that it does not handle the content of the data it represents. The markup tags only specify *how* the data should be displayed, e.g. <H1> for first level heading, instead of *what* the data represents. Consequently, it became obvious to researchers that, since HTML is directed towards human use of documents, the need for an application centred markup language was crucial.

HTML is an application of SGML. Because of this, some people thought the answer to the problems lay in using SGML for the purpose (Wüthrich, 1998). However, the problem with SGML is that it is almost too comprehensive in what it can do, and is therefore a very complex language to use. This was one of the biggest motivations for developing XML. It was a language more powerful than HTML and easier to handle and use than SGML, but with the same expressiveness.

3.2 Characteristics of XML

XML differs from HTML in the areas discussed by Bosak (1997). First, it does not have any predefined markup tags, allowing users to freely define their own tags for the specific needs of the applications. Second, XML can define document structures that can be nested

to any level of complexity. And third, XML documents can be associated with a DTD, which contains rules and constraints for the document structure. Hence, the user will have the opportunity to directly validate an XML document with respect to the specific DTD.

Relating to the discussion of Seligman and Rosenthal (2001), XML is content oriented. It efficiently separates the content from the presentation, leaving the presentation part for stylesheet languages such as XSL. This is done because it is not implicit that the content of a document is to be presented at all.

Another aspect of XML that deserves attention is that it allows us to build more powerful languages or improve old ones. One concrete example is the creation of XHTML (Pemberton et al. 2000). It is a redefinition of HTML 4, with the same functionality as HTML. The addition is the facility to use XML syntax to obtain greater flexibility.

An ingredient that potentially could contribute to spread the use of XML is the fact that XML is not that complicated. Wüthrich (1998) describes it as an XML file being a simple text file, and that the structure of an XML file is so simple that you can write one "by hand".

Other advantages discussed in the XML context are for example that it is an *open* standard, i.e. it is not designed by a corporation or research group for their specific needs. Another example is that XML's strict syntax makes it simpler to implement applications (Dimitrov, 2000).

According to Jansz (1998) the only disadvantage in using XML compared to SGML is that XML, despite its remarkable growth, still is a relatively young standard. This means that the techniques and tools already existing for SGML are still being developed for XML. Presently, very few standards exist.

3.3 XML Documents

XML describes a set of data objects called documents (Bray et al., 2000). These documents consist of some text representing the content of the document and markup tags with information about the content. The documents are organised as tree structures building nested hierarchies of elements. The elements are defined with the markup tags and may contain character data for processing. They might also have attributes associated with them composed of name-value pairs.

Even if XML is considered to be a very open standard language, it still has to conform to some sort of rules and restrictions. One way of formulating rules and restrictions for XML documents described below is called *well-formedness*.

3.3.1 Well-formed XML Documents

For a data object to be an XML document it has to be *well-formed*, as defined in Bray et al. (2000). To be well-formed it has to fulfil some criteria of how its structure is built up. Some of these criteria are described here. An XML document is well-formed if:

- It begins with an XML-declaration to identify it as an XML document.
- There is exactly one element called the *root*, or document element, no part of which appears in the content of any other element (Bray et al. 2000, p7).
- The other elements in the document, delimited by start- and end-tags, nest properly within each other, i.e. no overlaps are allowed.
- Every start-tag has a corresponding end-tag.
- The value of an element's attribute is enclosed in apostrophes or double quotes (Dimitrov, 2000).
- It meets all the well-formedness constraints given in Bray et al. (2000).
- Each of the parsed entities which is referenced directly or indirectly within the document is well-formed.

The meanings of these rules deserve to be explained. Firstly, when an XML document is written, it must include the XML declaration at the top:

```
<?xml version=”1.0”?>
```

This row declares that the document conforms to the defined version of xml, in this case version 1.0 (currently the most used version). The version attribute is required for the document to be well-formed.

The root is the element containing all the other elements. Because of this, it is important that only one root exists per document. Moreover, a root element may appear only once in a document.

An element’s content may contain other elements. But, each start-tag must be closed with its end-tag in the reverse order it was opened, i.e. if an element contains another element it must not be closed before the contained element is closed. This is an example of an in-correct nesting of elements:

```
<name>
<last_name>Doe<first_name>
</last_name>John</name>
</first_name>
```

For the example above to be well-formed the order of the nesting should be:

```
<name>
    <last_name>Doe</last_name>
    <first_name>John</first_name>
</name>
```

In HTML, it is permitted to for example use the tag to start a new paragraph and then not close it with the tag. This is not allowed in the stricter XML syntax. In XML, a start-tag, e.g. <name>, *must* have a corresponding end-tag, </name>.

Unlike HTML attributes, XML attributes have to be enclosed in either a pair of single quotes or a pair of double quotes, e.g. <person id=123> is not acceptable, while <person id='123'> and <person id="123"> are. Parsed entities are discussed later.

Figure 1 is a good example of a well-formed document. For a more detailed explanation about the well-formedness rules and constraints the reader is referred to Bray et al. (2000).

3.3.2 Document Structure

An XML document consists of text. The text in turn consists of intermingled *markup* and *character data*. Markup is defined by angle brackets (<…>) to efficiently depart it from the rest of the document text. All text in a document that is not markup is character data (Bray et al., 2000).

Figure 1 shows an example of a very simple XML document. At the top of the document is a so called *XML declaration*. This defines the version of XML used in the document, here 1.0. It also defines what type of encoding is used in the document, in this case UTF-8. The encoding decides what characters are valid to use in the document.

The first tag in the document, <my_movies>, is the root or alternatively the document element. The root contains all other elements. This can be seen, since the end-tag of <my_movies>, i.e. </my_movies>, ends the whole document.

The structure of an XML document can be defined as a tree with different levels. Bray et al. (2000) describe a way of representing relationships between all non-root elements. Figure 1 taken as an example, the <movie> is called the *parent* of <actor> and <actor> the *child* of <movie>, as the <actor> element belongs to the content of <movie>.

Each document has both a logical and physical structure, which must nest properly with each other for the document to be well-formed (Bray et al., 2000). The structures will be described below, each in turn.

my_movies.xml

```
<?xml version=”1.0” encoding=”UTF-8” ?>
<!DOCTYPE my_movies SYSTEM ”my_movies.dtd”>
<my_movies>
   <movie idno=”1”>
      <title>Braveheart</title>
      <category>drama</category>
      <screen_time unit_of_time=”minutes”>164</screen_time>
      <actor>Mel Gibson</actor>
      <actor>Sopie Marceau</actor>
      <on_loan status=”no”/>
   </movie>
   <movie idno=”2”>
      <title>The silence of the lambs</title>
      <category>thriller</category>
      <screen_time unit_of_time=”minutes”>114</screen_time>
      <actor>Anthony Hopkins</actor>
      <actor>Jodie Foster</actor>
      <on_loan status=”no”/>
   </movie>
   <movie idno=”3”>
      <title>Titanic</title>
      <category>drama</category>
      <screen_time unit_of_time=”minutes”>167</screen_time>
      <actor>Leonardo DiCaprio</actor>
      <actor>Kate Winslett</actor>
      <on_loan status=”yes”/>
   </movie>
   <movie idno=”4”>
      <title>The three musketeers</title>
      <category>comedy</category>
      <screen_time unit_of_time=”minutes”>99</screen_time>
      <actor>Charlie Sheen</actor>
      <actor>Kiefer Sutherland</actor>
      <actor>Chris O’Donnell</actor>
      <actor>Oliver Platt</actor>
      <on_loan status=”no”/>
   </movie>
.
.
.
</my_movies>
```

Figure 1: Example XML document

Logically a document consists of elements and their associated attributes. All elements have specific *types*, identified by the name of the start- and end-tag of the element. The attributes are defined by a name and a value. Together these name-value pairs form the *attribute specification* of an element.

Each XML document contains at least one element. The elements are indicated in the document by explicit markup. Elements begin with the definition of a *start-tag* and ends with an associated *end-tag*. The text between these tags is called the element's *content* (Bray et al. 2000). The content of an element can consist of character data, nested (sub-) elements or both.

It is not imperative that an element has content, though. There exist elements without any content at all. These are called *empty* elements. The empty element tags are different from regular tag definitions. The start and end tags are defined in one common tag.

The logical XML document structure can consist of other things as well. Among these are comments, CDATA sections and processing instructions three of the most common ones. *Comments* are sections of text that are not part of the document's character data. *CDATA sections* are used to enclose sections of data that would otherwise be considered to be markup. *Processing instructions* (PIs) are used to allow documents to contain specific instructions for the application that processes them. They are passed through the XML processor to the application (Bray et al., 2000).

The physical structure of an XML document is composed of so called entities. These are the physical storage units of the document (Bray et al., 2000). Entities are defined by *name*, and as opposed to elements, *all* entities have content. Dimitrov (2000) defines an entity as an abbreviation for a part of an XML document that is often used.

A document begins with a *root* or document entity, which contains the whole document (Bray et al., 2000). Unlike other entities, the document entity has no name. This entity serves as a starting point for the XML processor.

3.4 XML Semantics

In addition to well-formedness, the content of an XML document can be further constrained. This can be done with the help from a Document Type Definition, or DTD for short. The DTD could be said to provide a grammar for restricting the structure of an XML document. It provides rules for what could be modelled by the document, which elements that could be used and how they might be combined (Dimitrov, 2000). Documents can be associated with a specific DTD, which then serves as a semantic schema for the documents associated with it.

A document is considered *valid* if it has an associated DTD, and if the document complies with the constraints expressed in it (Bray et al, 2000). However, this validity is not of general nature. Chawathe (1999) emphasises the fact that the document is only valid with respect to *that* specific DTD.

Another important thing to mention is that it is not mandatory for an XML document to be associated with a DTD (Mecca et al., 1999). However, from the discussion of Mecca et al. (1999) it is possible to identify a minimal DTD for every XML document created, if so desired. Thus, all documents could be made valid just by conducting a simple syntactic check of the document.

To be able to process the structure of an XML document an XML Processor is needed. This processor is used to read XML documents and provide access to their content and structure (Bray et al., 2000, p 4). But to validate a document, a special kind of XML processor is needed. There are two kinds of processors, non-validating and validating. As might be expected, the validating processor is the one capable of validating the structure of a document (Bray et al., 2000, p 44).

There are however some disadvantages with DTDs. Since they were originally intended for document management, they provide inadequate support for modelling data. This means that some applications want to define rules that DTDs are not capable of handling.

Another problem is that DTDs use their own syntax for describing constraints on documents. This means an overhead when developing documents and DTDs, as the user is forced to learn two different languages.

In response to the problems of data modelling in XML, W3C created an XML Schema Working Group. The resulting standard recommendation was proposed in May 2001 (Fallside, 2001). The XML Schema language offers a new way of specifying the structure of an XML document. It solves many of the problems associated with DTDs (Erdmann and Studer, 2001). The plan is that XML Schema eventually will replace the XML-DTD language as the *de facto* standard schema language for XML (Dimitrov, 2000).

3.4.1 DTD Structure

DTDs are created from a schema language specifically provided for defining constraints and rules for the XML documents. The DTD language is not built from the XML language, but instead uses a totally different syntax.

my_movies.dtd

```
<!ELEMENT my_movies (movie+)>
<!ELEMENT movie (title, category, screen_time, actor*, on_loan)>
<!ATTLIST movie idno CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT category (#PCDATA)>
<!ELEMENT screen_time (#PCDATA)>
<!ATTLIST screen_time unit_of_time (minutes | hours) ”minutes”>
<!ELEMENT actor (#PCDATA)>
<!ELEMENT on_loan EMPTY>
<!ATTLIST on_loan status (no | yes) ”no”>
```

Figure 2: DTD for the my_movies document

A DTD is mainly composed of a combination of element type and attribute list declarations (Bray et al, 2000). Figure 2 gives an example of a DTD for the XML document in Figure 1. One important thing to notice in this context is that, as opposed to XML documents,

there is no concept of a root in a DTD. A document that conforms to a specific DTD can be rooted at any element in the DTD (Shanmugasundaram et al., 1999).

Element type declarations describe the allowed types of elements for a document associated to the DTD. These declarations constitute a finite set of different element types that may be used in the document. A declaration begins with <!ELEMENT *element_type_name…*>. If an associated document contains an element of a type not declared in the DTD, the document is not valid. However it might well be well-formed.

An element type declaration constrains the element's content, i.e. it defines what type(s) of content an element of that specific type is allowed to have. Different kinds of content can be defined: *element* content, *character data* content and *mixed* content (Bray et al., 2000).

If the allowed content is other (child) elements these are enumerated inside a parenthesis after the element type name. The number of possible occurrences of the child elements can be specified. To define this, the name of an enumerated child element is followed by one of the symbols in Table 1, as defined by Dimitrov (2000):

Symbol	**Meaning**
+	One or more times
*	Zero or more times
?	Zero or one time
<nothing>	Exactly once

Table 1: Classification of element occurrence

If the allowed content is character data, this is indicated with the #PCDATA keyword, which stands for *parsed character data*. This means that the element type has markup content which the XML processor should process (Chawathe, 1999). Element types defined with #PCDATA content cannot have any children.

An element type can also be defined to have mixed content. This means that an element of this type can have both text and child elements as content. With mixed content the types of

the child elements can be constrained, but not their order and number of occurrences (Bray et al., 2000). If an element type is defined with the ANY keyword as content, then it could contain child elements and/or parsed character data.

Elements can also have attributes associated with them. To declare these a*ttribute list declarations* are used. In Bray et al. (2000, p24) it is stated that attribute list declarations may be used to:

- Define the set of attributes pertaining to a given element type
- Establish type constraints for these attributes
- Provide default values

The declarations begin with <!ATTLIST… followed by the associated element type name. After that comes the attribute type name with its data type, possibly followed by a default value for the attribute type. The attribute list declarations are usually defined in direct succession to their element type.

Attribute types can be of three different kinds: string, identification, and enumeration, where enumerated attributes can take one of a listed set of values (Bray et al., 2000). The string attribute type is specified with the CDATA keyword. This type defines that the attribute can only have text values.

The identification attribute types constitute the ID/IDREF mechanism of XML, which can be used to restrict elements. The ID is an attribute used to uniquely identify an element, much like a primary key in a relational database table. The IDREF is an attribute used to reference elements, in about the same way as a foreign key is used in the relational model (Kappel et al., 1999).

The attribute list declaration also provides information about whether the presence of an attribute is required or not. There are three possible keywords to use in this context. #**REQUIRED** means that the attribute is mandatory, #**IMPLIED** states that no default

value is provided for the attribute, and #**FIXED** means that the instances of that attribute must always have the default value.

There are different ways to associate an XML document with a DTD. A common method is to declare in the document which DTD to be used. This is done with a *document type declaration*. The declaration is placed in the *prolog* of the document, i.e. in the beginning together with the XML declaration (see Figure 1). The document type declaration is the <!DOCTYPE…> tag together with the name of the root and a path to the specific DTD. If the DTD is locally stored the **SYSTEM** keyword is used to locate it, if it is a remote DTD the word **PUBLIC** is used.

3.4.2 XML Schema

W3C founded the XML Schema Working Group in response to the ever growing demands from both inside members and outside stakeholders (Dimitrov, 2000). The issue was a disappointment in the expressiveness of the DTD language, and the demand was to create a more sophisticated schema language for XML.

The work was initialised in February 1999 with the goal of finishing a schema standard as soon as possible. After more than two years work, a standard recommendation was announced on May 1 2001 (Fallside, 2001). It had then become a fully mature, stable standard and is now backed by all the 510 W3C member organisations.

The XML Schema standard recommendation is described in three different documents. The first, Part 0, is a primer (Fallside, 2001). It is a document in plain English that describes what schemas are, how they differ from DTDs, and how schemas are created using the XML Schema language. The second document, Part 1, proposes methods for describing the structure and constraining the contents of XML documents (Thompson et al., 2001). It also defines the rules concerning schema validation of documents. The third and last standard document, called Part 2, outlines facilities for defining data types to be used in XML Schema (Biron and Malhotra, 2001).

The XML Schema standard exceeds DTDs in a number of ways. First of all, unlike DTDs, XML Schema is defined in XML syntax. This means that a user familiar with handling XML documents should have no problem to understand the XML Schema language.

One of the most important improvements that XML Schema has over DTDs is the expressive power of XML schemas. The restricted capacity of DTDs makes them only sufficient to be used by certain types of applications. In contrast, XML Schema has many built in facilities missing in DTDs. For example, in contrast to the very poor set of data types supported in DTDs, XML Schema allows a large number of alternative data types to be defined, both for elements and attributes. The XML Schema standard has more than 40 built-in types defined (Dimitrov, 2000). It contains support for user-defined types, and can be used to define more powerful relationships and constraints than with DTDs. Furthermore, with XML Schema the user can define inheritance for both elements and attributes.

Dimitrov (2000) explains that XML schemas have native support for namespaces. This is a big advantage over DTDs, since they do not have any corresponding functionality. Namespaces are defined for elements to avoid naming conflicts in XML documents. It is not totally unusual that many tags in an XML document carry the same syntactic name. Since tags in XML do not carry explicit semantic meaning, two different elements (tags) with the same name will be in conflict with each other (Dimitrov, 2000). The elements that are specified in a document come from a certain vocabulary, i.e. a collection of allowable concepts. However, all elements in a document are not restricted to come from the same vocabulary. This will make the situation even worse.

As an example, consider the element <NAME> … </NAME>. This is a common concept, and might very well be used in different contexts. In XML, when only DTDs are used, this problem is solved by using the *Namespaces in XML* standard (Bray et al., 1999). Namespaces are used practically by adding a prefix to the element. Thus, if two different elements are declared as <NAME> … </NAME>, they can be separated by declaring them as <some_namespace:NAME> … </NAME> and <some_other_namespace:NAME> … </NAME>.

my_movies.xsd

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

    <xsd:element name="my_movies" type="My_MoviesType"/>

    <xsd:simpleType name=”My_Enum”>
        <restriction base=”xsd:string”>
                <enumeration value=”minutes”/>
                <enumeration value=”hours”/>
        </restriction>
    </xsd:simpleType>

    <xsd:complexType name="My_MoviesType">
        <xsd:sequence>
            <xsd:element ref="movie" maxOccurs="unbound"/>
        </xsd:sequence>
    </xsd:complexType>

    <xsd:element name="on_loan">
        <xsd:complexType>
            <xsd:attribute name="status" type="xsd:boolean" default="false"/>
        </xsd:complexType>
    </xsd:element>

    <xsd:element name="screen_time">
        <xsd:complexType>
            <xsd:simpleContent>
                <xsd:extension base="xsd:integer">
                    <xsd:attribute name="unit_of_time" type="My_Enum" default="minutes"/>
                </xsd:extension>
        </xsd:simpleContent>
    </xsd:complexType>
    </xsd:element>

    <xsd:element name="movie">
        <xsd:complexType>
            <xsd:sequence>
            <xsd:element name="title" type="xsd:string" maxOccurs="1"/>
            <xsd:element name="category" type="xsd:string" maxOccurs="1"/>
            <xsd:element ref="screen_time" maxOccurs="1"/>
            <xsd:element name="actor" type="xsd:string" maxOccurs="unbound"/>
                <xsd:element ref="on_loan" maxOccurs="1"/>
            </xsd:sequence>
        <xsd:attribute name="idno" type="xsd:ID" use="required"/>
        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

Figure 3: XML Schema for the my_movies document

Like DTDs, XML schemas are used to enforce structural constraints on XML documents. As mentioned before, the user has a considerable number of data types to aid in the structuring process. Some of the most common examples are booleans, strings, integers and decimals. These data types can be extended and inherited by using the *base* keyword.

Figure 3 gives an example of an XML schema defined for the *my_movies.xml* document in Figure 1. As can be seen, the schema defines elements and attributes like DTDs. However, as opposed to DTDs, types are specified for the elements and the attributes. Types can be of different kinds (Biron and Malhotra, 2001). *Simple types* are defined for elements that contain only text or numbers, but do not contain any subelements. *Complex types* are specified for elements that contain subelements and/or carry attributes (see Figure 3). Attributes always have simple types.

The user has the choice of specifying built-in types, e.g. boolean or integer, or define new types for the elements and attributes in the schema. Taking Figure 3 as an example, the *actor* element has a built-in string type, whereas the *unit_of_time* attribute has a user defined simple type. Special definitions have to be done for elements without subelements but with attributes, as with the *screen_time* element in Figure 3. The same goes for empty elements, such as the *on_loan* element.

3.5 XML and semistructured data

Semistructured data could be defined as data that is irregular or incomplete in nature (McHugh and Widom, 1999). The data might have missing attributes, multiple occurrences of the same attribute or attributes could perhaps be of different type depending on which part of the data it is associated with.

Semistructured data is *self-describing*. This means that the information about data that is normally associated with a schema is contained *within* the data itself (Buneman, 1997). In some cases the semistructured data does not have any schema at all, and in the cases where schemas do exist they only place very loose constraints on the data.

XML is closely related to semistructured data. Like semistructured data, XML data has irregular and flexible structure. It may change frequently and without notice. Because of this, much effort has been put into using graph based semistructured data models to model XML data. However, the mapping between semistructured data and XML is not a one-to-one correspondence. A considerable difference between the two is that XML data has to be ordered (Suciu, 1998).

One of the most known examples, which shows the similarities between semistructured data and XML, is the *Lore* project at Stanford. The project resulted in a complete database management system for semistructured data, called Lore (McHugh et al., 1997). Later, the system has been adapted to fit XML data.

3.6 Querying XML

The growing amount of information being stored, exchanged, and presented as XML has introduced new requirements from several communities. Businesses need new ways to enable their applications to perform more sophisticated searches through their XML data sources. The keyword based searches used by today's Web search engines are not sufficient for retrieving all the interesting information needed (Tian et al., 2001). This problem, along with other ones, has formed a demand for a query language for XML data.

For the exchange of information between businesses with the use of XML to be successful, facilities for storing, extracting, converting, transforming and integrating XML data need to be created (Fernandez and Suciu, 1998). All these issues assume the existence of a standard XML query language. This is something that has gained a lot of interest in the database community. As a result of this interest, leading database researchers from all around the world converged in a workshop in December 1998. The workshop was called *The Query Languages Workshop* (QL'98)[1]. The goal was to discuss the specifics of an XML query language. Several different approaches were presented and discussed, and

[1] Details about the workshop can be found at: http://www.w3.org/TandS/QL/QL98.

many of them have later turned into fully-fledged proposals for a query language standard. Some of these proposals are described in the following sub-section.

In the year after the QL'98, an ever increasing number of voices were raised, demanding an introduction of a generally agreed standardisation for an XML query language. As an answer to this, the W3C formed the XML Query Working Group in September 1999 (Chinwala et al., 2001). The aim was to establish requirements needed to be fulfilled by a standardised query language for XML. The first resulting Requirements Document was published in January 2000, and was updated in August the same year. The latest update was published in February 2001 (Chamberlin et al., 2001b). At the same time, database researchers in different places defined their own requirements and expectations of an XML query standard, e.g. Maier (1998) and Deutsch et al. (1999a). Although they inevitably differ in some opinions, they have a remarkable agreement in others, even with respect to the W3C requirements.

3.6.1 Query languages for XML

This section presents some examples of XML query language proposals. The goal is to give an insight into the XML query language area and a representative selection of languages is presented briefly. These languages are: Lorel, XML-QL, XML-GL, XQL, XSL, Quilt, and XQuery. In the following sub-section we will present the functionality of XSL in greater detail. The reason for this is that we believe XSL to be the most widely implemented and used query language to date.

All the example languages have been created by different communities. Some were designed by database research people while others were developed by the document community (Chinwala et al., 2001).

A simple classification of five selected languages has been done by Bonifati and Ceri (1999) to easily characterise and separate them. The classification has been made by dividing the languages into two broad categories, based on their characteristics and expressibility. The categories are called Class1 and 2. The Class 1 languages are those who

can only process simple queries over a single XML document at a time, whereas Class 2 languages are able to specify nested and multi-document queries. The languages reviewed by Bonifati and Ceri (1999) are:

- Lorel
- XML-QL
- XML-GL
- XSL
- XQL

Lorel was created by the database research group at Stanford University in California, USA (McHugh et al., 1997). The project called the "Lore Project" resulted in a complete database management system for semistructured data, and Lorel was the query language of Lore. Thus, Lorel was originally designed for querying semistructured data. However, later on it has been adapted to enable querying on XML data as well (Goldman et al., 1999). Since Lorel has a sophisticated query mechanism, through the use of powerful path expressions, Bonifati and Ceri (1999) have placed it in Class 2.

XML-QL is one of the most popular languages for XML querying, and is very commonly used as an example language when discussing the possibilities and obstacles for querying XML data. XML-QL was designed by database researchers at AT&T Labs, University of Pennsylvania, University of Washington and INRIA, France, and was submitted to the W3C in August 1998 (Deutsch et al., 1998). XML-QL has been designed by transferring techniques from query languages for semistructured data (Suciu, 1998). This was the first submitted proposal for an XML query language. The expressive power of XML-QL is very similar to Lorel, and it is therefore considered to be a Class 2 language.

XML-GL is a graphical query language for XML, with possibilities of specifying queries in a graphical user interface. It was developed at Politecnico di Milano in Italy (Ceri et al., 1999). XML-GL represents XML documents and DTDs as labelled graphs. The elements in the graphs are visually displayed, giving good feedback to the user. XML-GL has

powerful query facilities, but it lacks some of the most complex mechanisms. This has resulted in Bonifati and Ceri (1999) placing XML-GL somewhere in between Class 1 and Class 2.

XSL was originally developed as a language for manipulating and transforming the content of XML documents (Adler et al., 2000). The manipulation and transformation is done by matching patterns against elements in a document. XSL has been considered to have facilities that could be used for querying XML. However, due to its origin, the expressive power of XSL as a query language is relatively primitive. This means that XSL, unlike Lorel and XML-QL, lacks mechanisms for multi-document processing and joins on documents. Because of these shortages, XSL forms a good example of a Class 1 language.

XQL is a query language that originated from the document processing community, and was designed specifically for XML documents. A coalition of people from Microsoft, Texcel Inc. and webMethods Inc. cooperated and in September 1998 they proposed a simple and concise language for XML (Robie et al., 1998). One of the most important design goals for XQL is simplicity both in reading and writing, but this is at a cost of reduced expressive power. XQL can be seen as a natural extension to XSL (Robie et al., 1998). The language consists of XSL match patterns, which are used in the same way as in XSL. Due to this close resemblance to XSL, e.g. no nested querying or joins, Bonifati and Ceri (1999) have placed XQL in Class 1.

There also exist other languages, not participating in the classification of Bonifati and Ceri (1999), which have been proposed as candidates for an XML query language. Some examples of these languages are given below.

Quilt is the second newest proposed query language for XML. It has been created by Jonathan Robie from Texcel Inc., Don Chamberlin from IBM and Daniela Florescu from INRIA in France (Robie et al., 2000). Quilt was proposed in June 2000, and the design goals were to create an XML query language that constituted a perfect combination of the best ideas from all the earlier proposed languages. The idea was to adapt the borrowed features to form a new and fresh syntactic approach (Chinwala et al., 2001). Quilt could

not be considered to be document-oriented or database-oriented, since it has been influenced by so many different languages. It should rather be seen as a try to create a best-of-both worlds approach.

XQuery is the newest XML query language proposal. It is W3Cs own proposal, and is still under development. The design goal is for it to be widely applicable across all types of XML data sources (Chamberlin et al., 2001a). It has been derived from the Quilt language. XQuery is being designed to explicitly meet the requirements identified by the W3C XML Query Working Group (Chamberlin et al., 2001b). The aim is to make a small, easily implementable language in which queries are concise and easily understood.

It is worth to notice that, although languages like XML-QL, Lorel and Quilt have achieved wide acceptance in the XML society, no agreement on standard approach seems to be within reach yet. However, even though no query standard exists, the developers of more sophisticated languages, e.g. XML-QL and Lorel, have presented demo implementations of their query proposals[2]. On the other hand, to the best of our knowledge, no really usable implementations are available for these kinds of languages.

The language that we see currently has the best conditions for being used as an XML query language is XSL. The query facilities in XSL are not very sophisticated, but they are sufficient for simple selection and projection querying. In addition, XSL has been created as an integral part of the XML framework, by W3C itself. Also, it is the only proposed language that has been completely implemented, and proved to work well for processing XML data. Therefore, we consider it to be motivated to take a closer look at the XSL functionality, to get an understanding for its applicability and restrictions.

3.6.2 XSL functionality

The *eXtensible Stylesheet Language* (XSL) was created to enable the manipulation and transformation of XML documents. XML was designed to separate the content of the data from the presentation. XML in itself only handles the content of the data, the formatting

[2] The demos can be acquired from www.att.research.com/~mff and http://WWW-DB.Stanford.EDU/lore/.

and representation of it is covered by XSL. This is an important feature since it is not imperative that XML documents are created for presentation purposes. When discussing the use of XSL, an abstraction is often used that focuses on the structure of the document. The document is considered to be a *tree* (Clark, 1999).

XSL takes a tree, called the *source tree*, as input, and then transforms it into a *result tree* of intended structure. This result tree is thereafter formatted for presentation, according to the instructions from the designer (Adler et al., 2000). This process is done by associating the XML document or data with a specific XSL stylesheet. The document and stylesheet are processed by an XSL *processor*. The process could roughly be separated into two different parts. First, the transformation of XML, and then the formatting. These parts have evolved to become two detached languages.

The transforming part of XSL is called *XSLT*, which short for XSL Transformations (Clark, 1999). It takes a source XML tree and transforms it into a result tree. When creating the result tree, the structure of the source tree can be filtered and reordered, making the structure of the result tree very different from the structure of the source tree (Adler et al., 2000).

Transformations are done with the help from *stylesheets*. Every stylesheet contains a set of transformation rules, which are used on the source tree to create the result tree (Chawathe, 1999). In turn, each rule consists of two parts: a *pattern* and a *template*. When executing a rule the pattern is matched against the nodes in the source tree, and the template is initialised to create part of the result tree. This execution begins by instantiating the rule that matches the root element in the source tree, and proceeds until the whole result tree is created.

Figure 4 gives an example of a very simple XSL stylesheet for transforming the *my_movies.xml* document in Figure 1 into HTML for presentation on the Web. More precisely, it presents the title and screen time of all movies in Figure 1 as HTML.

The stylesheet begins, like the XML document in Figure 1, with an XML declaration. The next row has a stylesheet declaration. It also specifies the namespace that the stylesheet

conforms to. The third row specifies a template together with a matching criterion. This particular operation will match the template with the root of the source tree. The HTML transformation and presentation parts are then specified inside the template scope. The data for the presentation is accessed by looping through the whole XML document and filter out the values of the title and screen_time elements. To be able to use this stylesheet, another row has to be added to the header of the XML document in Figure 1:

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

The declaration defines that the type of data to be processed is text and that the stylesheet to be used is *movie_style.xsl*. When the matching process is finished, the data is presented in the result tree as specified.

movie_style.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
   <html>
   <body>
      <table border="1" bgcolor="yellow">
         <tr>
            <th>Title</th>
            <th>Screen Time</th>
         </tr>
         <xsl:for-each select="my_movies/movie" order-by="+ title">
         <tr>
            <td><xsl:value-of select="title"/></td>
            <td><xsl:value-of select="screen_time"/></td>
         </tr>
         </xsl:for-each>
      </table>
   </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 4: An Example XSL Stylesheet

The *XSL Formatting Objects* (XSL-FO), which form the second part of XSL, are concerned with the presentation part of the XSL process. The result tree is interpreted and

formatted to be presented in the way intended by the designer of the stylesheet. XSL-FO consist of an XML vocabulary for specifying formatting, and can be divided into a number of formatting object *classes*, each representing a particular kind of formatting behaviour (Adler et al., 2000).

If the discussion above is mapped to the example XML document and XSL stylesheet, the result of the processing will be:

Title	**Length**
Braveheart	164
The silence of the lambs	114
The three musketeers	99
Titanic	167

This shows a table in HTML style, containing only columns for the title and length of the movies in the example XML document, as defined by the stylesheet. Note that only the four explicitly defined elements in Figure 1 are shown in the example.

Although primarily used for the above mentioned presentation purposes, it has been considered that the XSL facilities could serve as a basis for an XML query language. The XSL Working Group believes that it could be constructive to see to the existing technologies of W3C when trying to establish standards for an XML query language. To them it is important to maximise the amount of reuse of technologies within the W3C (W3C XSL Working Group, 1998). Therefore the XSL Working Group has recommended that W3C should build an XML query language based on the pattern matching and transformation facilities of XSL. A serious proposal for an XSL based query language together with suggestions for suitable extensions to XSL is presented in Bosworth (1998).

When looking at the XSL syntax it is easy to realise that the pattern matching operations might as well be used for query processing. The select patterns retrieves specified nodes, similar to the SELECT clause in SQL. It is also possible to restrict the retrieval with filtering pattern qualifiers within brackets [ ]. This could be seen as a simple kind of SELECT-WHERE processing operation. The templates in the stylesheets used to create the

result tree could be compared to the tree creation facilities in more sophisticated XML query languages, e.g. the CONSTRUCT clause of XML-QL (Deutsch et al., 1998).

The XML document in Figure 1 and the XSL stylesheet in Figure 4 will be used to give an example of the discussion above. The stylesheet works as a query, retrieving the “title” and “screen_time” nodes from all movies. The stylesheet could easily be changed to further restrict the retrieval of values, e.g. by extending the 11th row with a condition:

<xsl:for-each select="my_movies/movie**[screen_time > 120]**" order-by="+ title">

The condition specifies that the screen time of a movie has to be more than two hours for it to be added in the result tree. The > is used instead of the > sign to avoid mix-ups with tag definitions. When applying the new stylesheet to the XML document, instead of showing all movies, this is all that will be shown:

Title	**Screen Time**
Braveheart	164
Titanic	167

The “order-by” keyword works as in SQL. In this case the ordering will be on “title”, and the + sign means that the ordering will be ascending, as opposed to descending (-).

3.7 XPath and XPointer

The pattern matching in the XSLT process is done by using the syntax of the *XML Path Language*, or XPath for short (Clark and DeRose, 1999). The primary purpose of the XPath language is to enable addressing of specific parts of an XML document. XPath uses a path notation for traversing through the hierarchical structure of an XML document, thereby the name.

Rather than working directly on the physical surface, XPath is operating on the logical structure of an XML document. It models the document as a tree of different kinds of nodes, e.g. element nodes, attribute nodes and text nodes (Clark and DeRose, 1999).

The key syntactic construct of XPath is the notion of *path expressions* (Chawathe, 1999). These expressions are used for the navigation through the XML tree. All expressions are created and evaluated in a certain *context*. This context is specified by defining a specific context node, which will be used as a basis for all the navigation in the tree. A set of nodes called the *context node list* along with a function library is also created, among other things.

The function library in XPath contains operations that can be used in the definition of path expressions. These functions can be used to determine the position of a specific node or to identify the root of the tree.

Even though XPath can be used satisfyingly in some areas, it becomes inefficient in other areas (Chawathe, 1999). As an answer to this, W3C formed a working group with the aim of proposing a new document addressing language standard. The new language is called *XML Pointer Language* (XPointer).

XPointer has been built on top of XPath, using the same syntax with path expressions and a function library. However, XPointer has extensions to XPath, which allows it to (DeRose et al., 2001a):

- Address points and ranges as well as whole nodes
- Locate information by string matching
- Use addressing expressions in URI references as fragment identifiers (after suitable escaping)

XPointer also extends XPath by defining additional functions to the function library (Clark and DeRose, 1999). This allows it to e.g. address parts of an XML document that are not well-formed, something that is not possible in XPath.

The W3C has also begun the work to propose a new standard recommendation of the XPath language (XPath 2.0). The aim is to improve the language to better align with

XPointer and the XML Schema Language. For more detailed information about goals and requirements of XPath 2.0 the reader is referred to Muench et al. (2001).

3.8 Related XML technologies

Due to its simplicity and extensibility XML is a very flexible language. Because of this flexibility, XML is today more and more used for other purposes than was originally intended. Although originally a document markup language, XML is now prompting an approach more focused on data exchange (Abiteboul, 1999). Businesses interested in exchanging information with each other encode their data in XML format before sending it, making XML serve as a kind of middleware mechanism.

XML has many related technologies suitable for use in the web and data exchange context. Some of these technologies are XLink, RDF and XForms.

The *XML Linking Language* (XLink) is a standard closely related to XPointer (DeRose et al., 2001b). The XLink language is used to create and describe links. The links are created with XML syntax inside XML documents. They specify relationships between so called resources. A *resource* is any addressable unit of information or service. Resources are addressed through a URI (Unified Resource Identifier).

One of the most common uses of XLink is for specifying hyperlinks. It could be used to create simple HTML hyperlinks, but it is also possible to extend the hyperlink functionality, making the hyperlinking more scalable and flexible (DeRose et al., 2001b).

The *Resource Description Framework* (RDF) has been created to enable encoding and exchange of structured metadata, i.e. data describing data on the web (Lassila and Swick, 1999). RDF provides a means for building up descriptions and facts about some topic. The syntax used is based on XML.

RDF is defined by the use of documents. Every RDF document could be seen as a group of statements that describe resources (Dimitrov, 2000). Resources are described by their

properties, and every property is associated with a type and a value. Values could be atomic, e.g. strings or numbers, or other resources.

XForms is W3C's response to the growing demand from web applications and electronic commerce solutions for better web forms with richer interactions (Dubinko et al., 2001). Forms are an important part of the web, and they continue to be the primary means for interactivity between web sites. However, the current design of web forms does not separate the purpose from the presentation of a form. XForms, in contrast, consist of separate sections that describe *what* the form does, and *how* it looks. This allows for flexible presentation options.

The practical use of XML today varies considerably from simple document processing to the above mentioned data interchange. This evolution leads to new requirements and demands on the use of XML (Rosenthal et al., 1999). Stakeholders from different communities, e.g. database researchers and commercial application developers, are all forming their specific expectations on the applicability of XML.

4 General data transformation problems

No matter what kind of solution EMW wants to consider, there are some general problems involved in a process where data is transformed between different representations.

Transforming data from one format into another is seldom a straightforward process (Abiteboul et al., 1999). The problems and obstacles that may appear tend to be recurrent in every transformation process. However, how hard these problems are to solve, and by what means, is often relative to the people conducting the transformation and the purpose of it.

The transformation between formats requires some kind of mapping mechanism or program, taking the source data and converting it to a structure appropriate for the new schema. The important thing in this process is the notion of *schema equivalence* (Atzeni and Torlone, 1995). Two expressions are equivalent if they yield the same observable results in all contexts of a language (Abadi, 1999). An ideal transformation will map all equivalent expressions of one schema to corresponding equivalent expressions in another. A transformation succeeding with this is called “equationally fully abstract” by Abadi (1999). This kind of mapping is guaranteed not to introduce any information leaks, i.e. it preserves the semantic significance. It will also preserve the integrity properties of the expressions, which is just as important.

As mentioned above, the transformation of data is a non-trivial process; it typically requires a considerable programming effort. The common approach for transforming data like this is to write a specific program for every translation task. Abiteboul et al. (1999) call this way of action for a “naive” translation process. A big problem of doing this is that the translation will be further complicated by numerous technical aspects that are not really relevant to the transformation process (Abiteboul et al., 1999). Moreover, as always when dealing with human intervention, there is a considerable risk of human mistakes, making

the process more error-prone, and in the end less preserving. Kappel et al. (2000) mention another problem. Since these mapping mechanisms often are hard-coded, it will be very difficult to maintain them in case of changes. This unnecessarily increases the time and effort needed to create these mechanisms.

The problems of transformations are not solely based on the deficient knowledge of the programmers. There are also difficulties associated with the process as such, and the data formats included in the process. Even more issues arise when the goal of the transformation is to integrate several formats into one common representation (Haas et al., 1999).

The problems that arise are mostly relative to the *schematic heterogeneity* of the formats involved (Haas et al., 1999). Data of different kind is represented differently depending on the schema, and the more differences in the schemas, the bigger the heterogeneity. The schema heterogeneities are often results of what Kappel et al. (1999) call *data model heterogeneity*, i.e. fundamental differences between concepts provided by different representations. These are differences in e.g. structuring, identification and relationship modelling.

A high degree of heterogeneity unavoidably leads to many mismatches. These can manifest themselves in varying ways, and many authors give examples. One of the most commonly described problems that can arise is presented in Elmasri and Navathe (2000, pp 540) as *naming conflicts*. This refers to conflicts in the way concepts are named and described in different schemas. Naming conflicts can be of two types: *synonyms* and *homonyms*. A synonym is when one and the same concept is named and described in different ways in two or more schemas. Homonyms occur when schemas use a common name to describe two different concepts.

Another problem is *type conflicts*, i.e. a concept could be modelled as an object or entity type in one representation and as an attribute in another, or data in one representation is modelled as metadata in the other (Elmasri and Navathe, 2000, p 541).

A third problem presented by Elmasri and Navathe (2000, p 541) is *domain conflicts*. This refers to a situation where a concept is named in the same way in different schemas, but is modelled to belong to different domains. An example is that a SSN is defined as an integer in one schema, while another schema represents it as a character string. A problem related to this is that, when mapping one structure into another, there is no guarantee that every type domain in the source representation has a correspondence in the target (Kappel et al., 1999).

Papakonstantinou et al. (1996) present two further sources of difficulties when dealing with transformations. *Schema evolution* means that the format and contents of a source may change over time, and it is important to reflect this in target representations, to avoid incompatibilities. *Structure irregularities* is another problem that refers to the fact that not all representations follow a regular structure, and this is a problem that the modeller has to address.

When dealing with transformations where many representations are to be transformed into one common representation, there are also other things to consider. It is very important to have a means for identifying equivalent objects and merge them to avoid unnecessary redundant representations in the common format. In addition, it is essential to be able to decide when two entities from different sources are the same, although not named in the same way (Abiteboul et al., 1999).

The literature discusses much about possible solutions to the above mentioned problems. Abiteboul et al. (1999) suggest a formal approach to create a clean abstraction of all the different formats in the transformation process, together with means for specifying correspondences between different kinds of data, and for the transformation itself. This will simplify the process, since the approach enables the creation of powerful tools that can automate big parts of the process. As a result, much of the problematic code handling could be minimised. Atzeni and Torlone (1995) follow the same track by proposing a meta model for handling the transformation process and to keep order of different data formats.

Both Kappel et al. (1999) and Elmasri and Navathe (2000) present the same solution to the heterogeneity problems. The solution is to adapt one schema to better suit the representation of the other. However, this will decrease the *autonomy* of data assumed when transforming between formats (Kappel et al., 1999).

It is a general requirement that more and better tools and mechanisms to aid in the transformation process should be developed. More and more such tools are now being created, by Microsoft among others (Bernstein and Bergstraesser, 1999).

According to Hart et al. (1994) a prerequisite for successful transformation is to have considerable *domain expertise*. Here domain expertise does not only refer to expert knowledge of one domain, but rather knowledge exceeding several domain limits. Only by having this expertise, it is possible to efficiently build the necessary tools needed in the transformation process.

5 Problem

5.1 Motivation

The problems at EMW are associated with the management of data and retrieval of information. However, the level of detail on the problems is very low, which makes it hard to estimate their nature and scope. To be able to even speculate on possible solutions to the problems, it is necessary to more precisely specify the needs of EMW.

To be able to give EMW an answer to if XML lives up to their expectations, a straight-forward approach is to find a way to identify the parts of the XML framework that address the problems of EMW, and thereafter evaluate XML's suitability for solving existing problems at EMW. Here the XML framework is referred to as the XML core standard and its related technologies, as defined by the W3C.

As the problems of EMW, even though still defined on a very abstract level, seem to be general in nature, they are used as an integral part of the problem. Also, since an approach towards a solution has already been analysed, the results and experiences from that analysis are used. The results are relevant to consider, since a possible adoption of XML into the company will be done with the same purpose as was intended with the solution presented in that project. This gives more perspective to the problem.

5.2 Problem statement

The main aim of this project is to analyse to what extent XML can be used as a means for satisfying the requirements and needs of EMW, in comparison to the solution based on relational technology that has already been considered.

5.3 Objectives

To find answers to the problem above it is important to establish the work that is needed in order to get there. It is important to collect information from EMW, to be able to describe the current problematic situation. The gathered information is needed to specify the requirements and desires of EMW that have to be addressed. In addition, relevant parts of the XML framework are necessary to identify; to be able to determine how well XML could be used for the problems identified at EMW.

The following objectives have to be realised:

i) Get an understanding for the current situation in EMW's work and the problems that the company is facing.

ii) Specify the requirements and expectations that the EMW personnel have in order to solve the company's problems.

iii) Make an analysis of XML and its related technologies in order to identify problems and possibilities of adopting XML to the problems of EMW, in comparison to the earlier considered solution based on relational technology.

An important thing to notice here is that there is yet no universally established opinion on what XML really covers, i.e. the scope of the XML framework is a matter for debate.

5.4 Method

i) To get the necessary understanding for the current situation at EMW, elite interviews are performed, i.e. unstructured interviews on key technical staff, key staff meaning members of the staff directly working in the problem area. The employees that are interviewed are accessible staff members of EMW. Additionally, analyses of some example data structures are conducted. The reasons for interviewing the employees are because their opinions of the work they perform are verified

immediately, and it serves as a good complement to the analyses of the data, since some data is classified and therefore not accessible in this project.

ii) Summarise the information acquired from interviews and from analysing the data and use this to specify the requirements and expectations of EMW.

iii) Perform an analysis of XML in order to identify relevant parts and define the characteristics of the XML framework in the EMW problem context, in comparison to the solution that has been outlined earlier.

6 EMW's requirements and expectations

To be able to approach XML in the areas of interest from the view of EMW it is important to clarify which these areas are. Therefore interviews have been performed on key technical staff in the company to sort out their problems. Our aim with this has been that it would give a picture of the current working situation, and thereby provide information about *why* the problems are problems, not just *which* problems the personnel at EMW has.

To acquire the above mentioned understanding, two employees, actively involved in the radar communication work, were interviewed. The two employees are the only ones with sufficient context knowledge. Also, an insight into the structure of files created in the simulation has helped to make the study more complete. These interviews and file analyses have resulted in a description of the situation as it is, described in section 6.1, together with requirements and desiderata in the problem context, summarised in 6.2.

6.1 Current situation

As described in section 2.2, the data in focus is stored in big log-files. A file contains data about the communication flow between radar components, and it is built from hierarchical data structures. The sizes of files are many Gigabytes, and occasionally even Terabytes. The details about the communication and creation of files during flights, along with a description of the file structures, will be presented in the following subsections. When files have been stored they are available for analysis, and the current approach for analysing data is described in section 6.1.3. Last, problems with the approach on file structure and filtering are identified.

6.1.1 Internal radar communication

The situation of interest is communication scenarios. The communication could take place in real flights or simulated ones. In the real scenarios the communication takes place either in a real flight test or even with operational equipment in real use. In both cases, the equipment is located inside an aircraft. In cases where the scenarios are simulated, the simulation is performed on internal network architectures. The architectures consist of computers connected to each other, simulating the architecture of different hardware components in the radar systems of the aeroplanes. Since simulated flights are the basis for testing at EMW, the focus from here on will be on simulation.

The information about what happens during the communication scenarios is contained within *messages* sent on connections between hosts. These messages are sent according to the specific situation at hand. To name a few examples, in some scenarios they could be sent on a timely basis, and in others the sending of a message might be triggered by a certain event in the system.

The messages are sent to inform about state changes in some component. These changes consist of the altering of some attribute value(s), representing the internal structure of the component. The values are stored in messages before sending them to relevant destinations. A more exhaustive discussion about the relationship between messages and attribute values is presented later in this section.

When data describing the state of the components is sent on a network, it will have a binary form when arriving at the receiver. To make data comprehensive, it is structured according to some formerly agreed pattern. Due to the decoding mechanism described below, the receiver does not have any problem interpreting information. The patterns in the communication scenarios are very often defined in CORBA IDL (OMG, 2001). Therefore, pattern structures will hereafter be assumed to be built from IDL.

To clarify the discussion above, an example will be given. Take two components, host1 and host2 (see Figure 5). If the state of the internal structure of host1 is changed, for example, if one or more attribute values have been altered, it may be decided that this information is essential to acquire for another component, e.g. host2. The application of host1 then begins an encoding procedure. This procedure involves transforming the pattern structure into some specified programming language representation, This is done with the help from an "IDL Parser", which is an internal transformation mechanism at EMW. The resulting language could differ depending on what is specified in the encoding. Supported languages are e.g. Ada, C, C++ and Java. Since the applications do not use any universally agreed language, it becomes motivated to use a language like IDL for the purpose (OMG, 2001).

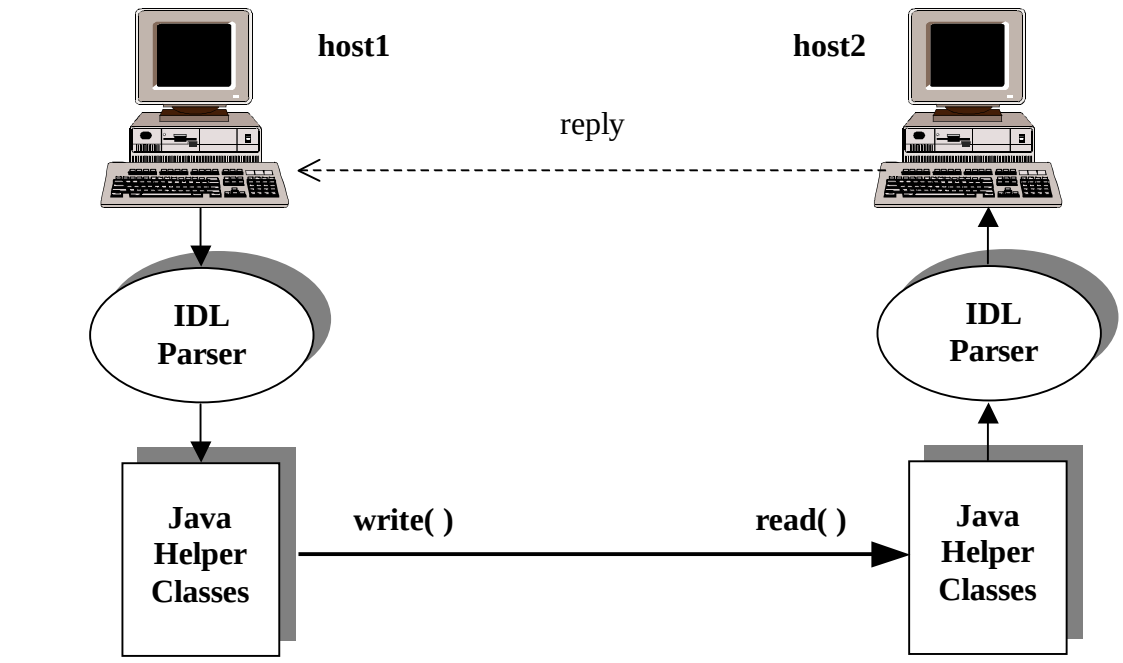


Figure 5: Internal communication model

Taking Java as an example, the processing consists of creating classes to help conclude the encoding process, as can be seen in Figure 5. Apart from classes for the message and all the attributes in it, a helper class for all classes is also created. The attribute classes contain the structures and values in their object-oriented form. The helper classes contain two very

useful operations, *read()* and *write()*, which are used in the transmission work. The write() operations transform all the attribute values to byte form, to be suited for transmission on the network.

When the whole message has been encoded and sent on the network, it is time for host2 to start processing. When the message arrives, host2 checks a message identifier to determine what type of message that has been sent. This is because components can send different kinds of messages between each other. The message identifier, abbreviated id, is found in a specific header that is attached to every sent message. The header concept is described in detail below.

The component uses the associated read() operations in its helper classes to decode the byte strings and extract the information. At the same time it sends a confirmation message ('reply') to host1, to inform that the message was received correctly (see Figure 5). The transmission procedure is then finished.

The above described procedure could be compared to object-orientation. While the pattern structures for all message types act as class definitions, the specific messages play the role of objects or instances of those patterns. The objects contain attributes, which in turn contain snapshot information about the states of the objects, much like the functionality of variables in object-oriented languages.

The messages are sent bit by bit on different network *buses*. These buses conform to specific *network protocols*. The communication will proceed according to the rules of the specific protocol. Since the communication between components is founded on an application based level, the protocols that are most often discussed in the context of EMW are *application-oriented* protocols (Halsall, 1996, pp 13-18), i.e. protocols adjusted for the specific needs of the EMW applications. Thus, when discussing protocols on a level more oriented at the management and processing of data, the term "application protocol" will be used to identify the type of protocols used. In contrast, the concept "network protocol" will be used when referring to protocols used to guide physical network communication traffic. In other words, protocols are discussed in two different contexts throughout the text.

A common network protocol standard used for the communication between components in the scenarios is regular *Ethernet* (Halsall, 1996, p 285), where the communication proceeds with media sensing and collision detection. Messages in the communication are split up and sent in many frames on an Ethernet bus.

Another commonly used standard is called *Mil-Std-1553*, or just 1553 for short (James and Honegger, 1996). 1553 was introduced in 1972, and it has been developed specifically for military avionics applications[3]. The development of 1553 became an answer to the need of new techniques for the transmission of information in aircrafts.

The notion of time is an important issue in this context. The time, at which messages are sent, is synchronised into certain intervals, called *intis* (integration intervals). This means that all messages sent during a specific inti will be stamped with the timing value of that inti. Hence, messages can be grouped and classified with respect to time. This possibility is useful when later analysing the data, searching for certain patterns in it. Information about what inti a message has been synchronised into can be contained within an attribute inside the body of a message. Attributes and message bodies are described later in this section.

Every application protocol has a set of data types defined for it. These types can be seen as a framework for how messages can be logically structured. The types defined for a protocol are mostly based on types from well established languages, e.g. C or CORBA IDL (as in Figure 5). Figure 6 shows an example of a set of types within a module that could be specified for a specific protocol, even though a real protocol might have many times more types defined.

Every message is defined to be associated with a *specific* application protocol. The relationship between protocols and messages is that every protocol can have a set of different message types defined for it. In turn, the specific *type* of a message is determined by one of the data types defined for the associated application protocol. The number of

[3] More details about the development of Mil-Std-1553 can be found at http://www.1553.com

different message types defined for a specific protocol can vary, but according to the EMW staff a mean number of 50 types per protocol could be assumed.

The significance of the protocol definitions is not that big in the context of message types, in that message types might have a high degree of resemblance to one another, even though they might be defined for different protocols. This means that the structures of different message types all have some mutual characteristics. The question is then why different network protocols are defined and used for the communication. One of the biggest reasons for this is due to legacy. As mentioned above, 1553 was introduced already in 1972, and much has happened in the market since then. The desire to modernise combined with a reluctance to totally abandon already implemented architectures made a solution with different network protocols attractive.

As stated above, the messages specified for a specific application protocol are of certain types. This means that the message definitions for a specific protocol will associate a message type with a certain application-oriented protocol data type. The types specified for an application protocol could be of two kinds: simple types and complex types. Well known examples of simple data types are: *long*, *boolean* and *char*. The simple types can be defined independent of any other types, making it very easy to access the data. However, simple types can also be in the content of complex types (see the Simple_Struct type definition in Figure 6). The complex types can even contain other complex types, which in turn hold their own complex types and so on, forming very deep nested hierarchies.

In the application protocols, the atomic values that need to be accessed during analysis are numeric data contained within the simple attributes of the message. The complex types are called "holders", since their instances hold other complex attributes and/or atomic valued attributes. This kind of hierarchical build-up could be compared to object-oriented class hierarchies. Complex types can be of different kinds (OMG, 2001):

- array
- struct
- sequence
- union

These are defined, each in turn, below.

An easy-to-understand type is the conventional *array*. This is just a holder of a number of elements, with a static number of storage places. An array definition is found in Figure 6, inside the False_Struct type definition.

A *struct* is a type holding a number of attributes of possibly different types, building a class-like structure. It reminds of an array, in that it contains a static number of elements. The big difference is that a struct has no restrictions on storing elements of specific types. Figure 6 gives some examples of struct definitions.

The *sequence* concept is referred to as a kind of dynamical array type. The places for storage are not pre-specified, and the size of the sequence can be managed on the fly. A special case of sequence is *string*, which potentially is a sequence containing only chars.

The sequence can at compile-time be defined to have a maximum number of elements, the maximum can be defined to be infinity, or it can just be defined to be an empty sequence (OMG, 2001). The size of the sequence is determined at run-time. This makes it harder to handle than the earlier described types, i.e. array and struct, since those types are completely specified at compile-time.

There is also a special kind of complex type, which is slightly different from the other ones described above. This type is called a *union*, and its structure is somewhat similar to that of a struct, in that it is a type that can hold attributes of other types. The IDL union could be said to be a combination of C union and switch statements (OMG, 2001). However, the unique thing about unions is that the type and value that they hold are not completely

specified at compile-time. Instead it has some possible attributes in its content in a *case* structure.

types.idl

```
module Types {

        typedef boolean My_Bool;

        typedef struct Simple_Struct {
                boolean     B_value;
                long        L_value;
                char        C_value;
        } S_Struct;

        typedef struct True_Struct {
                long        L_value;
                short       S_value;
                double      D_value;
                Types::Simple_Struct  S1;
        } T_Struct;

        typedef struct False_Struct {
                char        Char_Array[ 10 ];
        } F_Struct;

        typedef union Bool_Union switch ( Types::My_Bool ) {

                case TRUE:
                        Types::T_Struct  T_value;

                case FALSE:
                        Types::F_Struct  F_value;

        } B_Union;

};
```

Figure 6: Example type definitions for a protocol

One example of a union definition can be taken from Figure 6. Directly following the union definition is a selector called a *switch*. This switch has a parameter associated with it, called the *discriminator*, which controls the current value for the union. In Figure 6 the

discriminator is a boolean value. If the value is true the TRUE case will be executed, and the FALSE case will be disregarded. If the discriminator has a false value the opposite will happen. Comparing this to a tree structure, the switch will indicate on which branch to continue when passing the union. The specific value of the discriminator will not be known until runtime. As with the sequence, this makes the union a difficult type to handle.

Each message of a specific type is structured as follows; it is identified by a *header* containing necessary information about that specific message instance, followed by its data content, called its *body*. Figure 7 depicts a possible message header structure. The header information is contained within a Protocol module, associating the message's type with a specific application protocol.

protocol.idl

```
module Protocol {

    // Message-header

    struct Message_Header {
        unsigned long     Size_In_Bytes;
        unsigned long     Time;
        string            Message_Identity;
    };

};
```

Figure 7: Example of a message header definition

The size of the header is determined by the associated protocol[4]. The size is static for all the headers, and the headers associated with a specific protocol all have the same structure. This makes it easy to identify where the message data is stored, i.e. where the search is going to be done later when looking for atomic attribute values. This is further described in section 6.1.3.

[4] The *unsigned* keyword specifies that the value of the attribute must not be negative

The header specifies that a message has a certain size in bytes (Size_In_Bytes), as can be seen in Figure 7. This is important since messages in one file can be from different application protocols and be of different types, and therefore can vary considerably. This means that the size attribute is essential to inform about where the content of a message begins and ends, which will later be important when logging messages and filtering out information.

The message header can also possibly contain information about the time at which the message was sent on the network (see Figure 7), depending on the purpose of the communication. The message identity attribute in Figure 7 is a string for identifying the specific message type.

messages.idl

```
module Message {

    // Simple message containing a struct

    module Simple_Message_Type {

        typedef Types::S_Struct Msg;

        const string Message_Identity = "Message_Type_1";
    };

    // Complex message containing a union

    module Complex_Message_Type {

        typedef Types::B_Union Msg;

        const string Message_Identity = "Message_Type_2";
    };

};
```

Figure 8: Example message type definitions

When a radar component sends information over the network to another component, a message of a specific message type will be created and sent. This message gives a snapshot of the component's state. The snapshot information is placed in the body of the message.

As opposed to the headers, the bodies of messages of different types are structured in separate ways, even for one and the same application protocol. The body is structured according to the pattern specified for the specific message type. Figure 8 shows an example of two different message types possibly specified for a specific application protocol[5]. The simple message is associated with an S_Struct type, as defined in Figure 6, and a specified message identity, identifying the type of the message. The complex message has a B_Union as its type, also defined in Figure 6. This message type has an identifying attribute specified too.

Even if the types of the messages are defined to be single, the structure of messages still tends to be very complex. Take Figure 6 again, what if the Simple_Struct would be redefined to contain e.g. a union type? Then a message type associated with the True_Struct would suddenly have the structure of a struct containing a struct that contains a union. In this way extremely big nested hierarchies can be built from the definition of a single message type.

6.1.2 Data collection

There are several possible strategies for getting the information from the messages passed on the network, i.e. to collect interesting data. This is done by recording the collected data and store it in logs. Which recording equipment that is used for the purpose depends on the conditions for the communication, e.g. which network protocol that is being used.

One recording strategy is to use a software program called a *sniffer*. This sniffer is a mechanism that explicitly goes in and monitors the network to await passing messages, and when it detects them coming, it collects them.

[5] The *Msg* keyword is in this case an internal identifying name for messages.

There are also other strategies for collecting and logging data from messages sent on the network. One example is to have a double copy sending mechanism, i.e. when sending a message to its destination, it is also automatically sent to the recording equipment for processing.

Since the recording equipment does not have any information about the structure of the data in the messages, the raw data is simply copied directly. When message data has been collected it is wrapped into byte packages followed by streaming it down into a log-file. When the process is finished and all the packages are contained in the file, they are stored on disc.

As stated above, the log-files contain message data from the communication scenarios. The files consist of a sequential succession of message packages, all with a header-body structure of byte data. The data stored in the files is structured just as it was when sent on the network, i.e. no intermediate conversion is made. The header-body package of one message is immediately followed by another package, building a sequence in the file. This is shown in Figure 9.

Different parts of a file may differ in structure, e.g. a file could potentially contain messages from different application protocols, making the structure of the file varying. Taking Figure 9 as an example, the first and second header-body packages could be messages of different types, making them different in structure although stored in the same file.

The positioning of different message packages is determined by the size of the preceding package. More concretely, the header of every package contains data about the specific message, e.g. which protocol the message is associated with, the identity of the message etc. The header also contains a length attribute (see Figure 9) informing about the length of the body associated with that specific header. Apart from being used to specify where the data is located for a message, the length attribute shows where the header of the succeeding message is located. This is because the header of the succeeding message follows directly after the end of the body (see Figure 9). This way of structuring data is working due to the

fact that headers of one and the same message type all have the same predefined structure and size. Consequently, it is only necessary to get continuous information about the bodies.

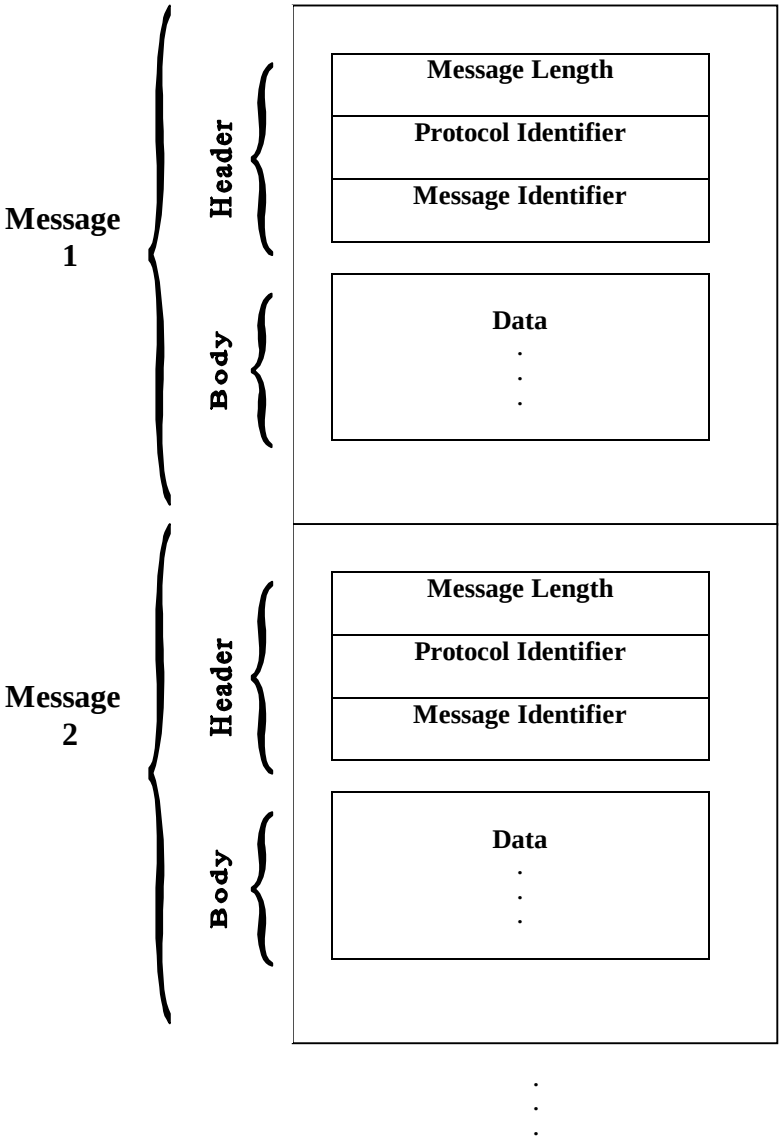


Figure 9: Example sequential structure of a log-file

As explained above, the length of the body is determined by the type of the specific message. Although every message only has one specific type, the lengths of the message bodies tend to be very big. This is due to the hierarchies built from the nesting of different

data types. It can straightforwardly be realised that very deep hierarchies are needed to form messages that together make up file content of many Gigabytes.

The number of files created during the collection of data in the communication varies. There could be one file per logging scenario, but this is not a restriction. More than one file can be created depending on the context in which the communication is conducted.

When the data collection is finished, the files are stored in archives on an internal network at EMW. There they are stored until needed for analysis.

6.1.3 Search and filtering for analysis

When the files from the data collection process have been created and stored they are ready to be analysed. This is done with the help from analysis tools, e.g. Matlab. To be able to fully utilise the analysing facilities in the tools, all relevant information in the files will need to be accessed. The information is contained within the atomic values of the simple attributes. Thus, a mechanism for identifying and access these values has to be used.

The current approach for filtering out information from the files is based on sequential search with the help from index tables. To be able to access the interesting information, filtration criteria need to be specified. This is done by executing a C-program, into which the analyst manually has to state the criteria. The program then works as a filter on behalf of the analyst. Moreover, if the analyst wants to make another, possibly very different, filtration, he/she must do the whole process all over again.

The whole filtering process is depicted in Figure 10. As data that fulfil the criteria of the search is found, it is retrieved to the filtering program, sorted and stored in a file for only filtered data, as shown in Figure 10. When all data has been scanned, the whole set of values that have matched the filtering conditions is stored in the new file generated by the filtering program. The data is thereafter sent to the analysis tool (see Figure 10). One of the most commonly used analysis tools at EMW is Matlab. Therefore Matlab will hereafter be used as an example tool for explanation purposes.

When the new file has been generated, the analysis tool uses a new C program for further filtration purposes. The tool can now specify new filtering criteria and use this program by function calls for retrieving information from the filtered file (see Figure 10). The values searched this time are the atomic values of the attributes. When values are found which fulfil the conditions, they are mapped to Matlab's internal representation. There they are stored in variables and placed in a matrix for analysis.

The mapping between the original types of the filtered data and the internal types of Matlab is often not a one-to-one correspondence. This is solved by using more universally defined types in Matlab as target types, e.g. all numeric values, whatever the type, will be stored as values of a universal numeric type in Matlab.

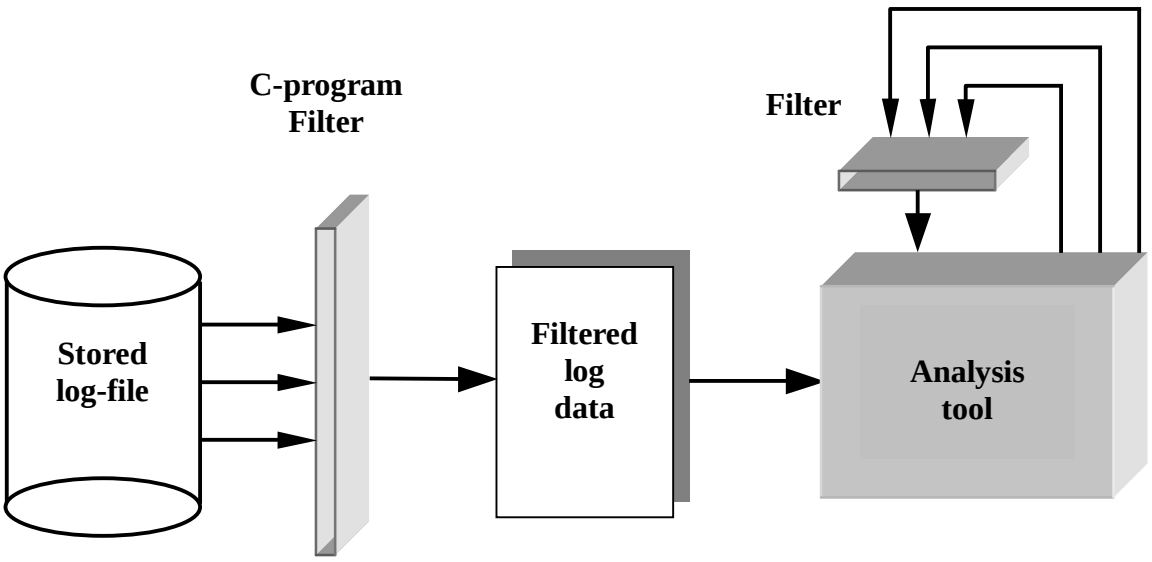


Figure 10: Filtering process

The process described above defines two separate filtering mechanisms, as can be seen in Figure 10. The first C-program makes a coarse filtration on the data stored in the log, and the other takes out the interesting atomic values. The big difference between the two is that the second filtering is done internally from the analysis tool.

One example of a question for the first filtering mechanism is to store the information about all messages logged during a certain inti-period, e.g. "Retrieve all messages between inti 5 and 10". Another possibility is to retrieve all messages of the same type stored in a file.

The questions asked on the generated file will be more specific, like "Which aeroplanes had a speed larger than 100 mph during the specified inti-period?". Another possible filtration is to find specific values in the same context, but which are stored in different files, and compare these against each other.

When executing the filtration criteria on a target file, a sequential search is conducted to find values that match the conditions in the filtration. To be able to do this as straight-forwardly as possible it is important that the searching mechanism is informed about where to start the search. This is not a difficult task considering all the message headers of one type have the same structure. As has been mentioned before, because every header is static it is easy to know where the message data begins, i.e. where the search is to commence.

When scanning the file the filtering program is looking for attribute values corresponding to the filtering criteria. This is done by gradually working through the file. When an attribute is found, it is checked and then the search continues. If the value of the attribute is a match, it is stored by the program in a *new* file.

As data is purely binary, it is imperative for efficient search to have a notion about where in the file the different attributes are stored, i.e. information about the order of attributes for every message type should be stored somewhere. The search is further complicated by the fact that there are many different data types and that all application protocols have their own representation, e.g. an integer does not necessarily take up the same amount of space in two different application protocols.

To solve the above mentioned problems, *index tables* are used. Figure 11 gives an illustration over information needed in the process of creating indexes. Every protocol has several different message types and each of these types has its own index structure

representing its specific message structure. The index tables contain information about the structural order of attributes and the size in bytes that each attribute takes up for the associated message type. This makes the retrieval of data faster, as the attribute values will be matched according to their position. Figure 11a gives an example of a very simple message type with its associated attributes. Along with the enumeration of attributes follows information about the space in bytes that every attribute takes up. Notice that a string takes one byte per character, plus 5 bytes to store the string length.

a) Message type structure and size

AttrName	Data type	Size
header:		
size	**long**	**4 bytes**
id	**string**	**5 bytes + string.length()**
body:		
a_number	**long**	**4 bytes**
comment	**string**	**5 bytes + string.length()**

b) Example_Message

size	**'37'**
id	**'message1'**
a_number	**'10'**
comment	**'test_string'**

Figure 11: Example of required indexing information. a) Message type with associated attributes. b) Message instance with attribute data.

As can be seen in Figure 11, the example message type has four attributes; two in the header and two in the body. The *size* and *id* attributes in the header have the same significance as earlier, i.e. the size attribute represents the size of a message instance in

bytes and the id identifies the type of the message. The attributes in the body are examples of a long integer and a common string, respectively.

In Figure 11b an example of a message containing data is shown. The first value indicates the total size of the message, whereas the second specifies that this is a message of type ”message1”. The body has also been populated, with dummy values.

The next step is to calculate the index values for all attributes. These values are then stored in the index table as pointers to specific positions in the file where the associated message instance is stored. When calculating these values, two approaches can be taken. Either the positions of the attributes in the header are taken into account, and the positions of the body attributes are just added after them. Or the header is simply disregarded, and the index values will only be stored for the body. For the data in Figure 11b this will mean that the *comment* attribute in the body could potentially have two distinct index values. If the header is counted the index for comment will be: 4 (size) + 5 (id) + 8 (id length) + 4 (a_number) = 21. On the other hand, if the header is disregarded, the index will be 4, since only the values of a_number need to be considered.

The whole process of filtering out interesting information is quite time consuming. This is because of the large amount of data stored in the logs. It is not unusual that the process takes several hours. The filtering is therefore often performed when resources are not needed elsewhere, e.g. during nights.

6.1.4 Problems with current approach

At the time being EMW has some problems with the representation and manipulation of their data. As mentioned above, when interesting information is to be filtered out for analysis, index tables are used to locate the searched value in the file content. The problem is that the search and filtering mechanisms are created for a static representation of data, i.e. that all the details about the message structures in files are known beforehand. However, the structure of files is more flexible and unpredictable, i.e. some parts of the file are not determined beforehand, due to the dynamic characteristics of e.g. unions and

sequences. When manipulating a simple structure, it is only to check the order with the indexes. However, when complex types such as a union or a sequence exist, this indexing approach encounters problems. There is no way to predetermine the structural details of such types. Since the index positions of all attributes are dependent on how much space the preceding attribute takes up, this indexing mechanism will be very hard to generate and use efficiently.

To clarify the discussion above, an example will be provided. Since the union is one of the most difficult types to handle when searching, it will be used in the example. The example union has a boolean discriminator. The case structure of the union is as follows: if the discriminator is true, then the union will take the value of an integer called *i*. If it is false, the value will be taken from a boolean called *b*. Thus, the stream of bytes stored in a file will vary depending on the discriminator.

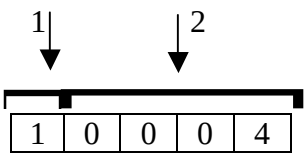
Assume that the discriminator is true and that the state of integer i is 4 when stored. This will make the structure in the file look as follows:

1	0	0	0	4

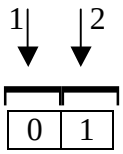
The first byte position defines that the discriminator is true and the next four shows that it is an integer with the value 4. If the discriminator is false instead, the value of boolean b will be true. This will give the following structure:

0	1

The index mechanism for these kinds of structures cannot be used as with simple types. This is because it is impossible to conduct the search in only one operation. To be able to retrieve the value of the union at all, it is first necessary to check the value of the discriminator. This will indicate what type of data that follows after. In the first example this will imply that the search would be performed in a way similar to this:



The 1) indicates the search for the value of the discriminator, while the 2) is the operation performed when retrieving the value of the integer. This process is needed, since the union will take up different amounts of space in a file depending on its current value. Compare the above byte string to the one that will be stored for the boolean b:



To somehow overcome the problem described above, a kind of nested indexing mechanism has been used. This works as follows: when an attribute with a complex structure is found with the help from the first index table, another table is used. That table is specific for that data type and will aid the analysis mechanism in identifying the searched value.

A new problem with the nested mechanism is that the searches tend to be complex, because of the increasing number of different index values that need to be handled at the same time. Furthermore, due to all the different protocols, all with their own message types and supported data types, the number of different index tables will be very large.

Yet another problem, based on the fact that complex types are involved, is that the current approach for generating index tables at EMW will get less automated. Since the generation of indexes is a very extensive process, as much of the work as possible has been automated. This automation tends to get much more difficult to create when not having access to the message structures beforehand, for obvious reasons.

EMW is, with respect to the discussion above, interested in finding a new way to solve the problems they currently have with the representation and manipulation of logged data. Of course they have their own requirements and desires concerning what such a solution should provide for them. The representative key staff has been interviewed about this, and the results of these interviews are presented below.

6.2 Requirements and Desiderata

When initially discussing the requirements of EMW with the key staff the focus seemed to be on efficiency characteristics solely. The EMW employees wanted to have a new way to efficiently represent their log data; they wanted efficient storage and optimised query processing with short response times. However, after internal discussions between members of the staff, they realised that the efficiency requirements only were secondary. The direction of the requirements was therefore changed to more elementary properties of the data.

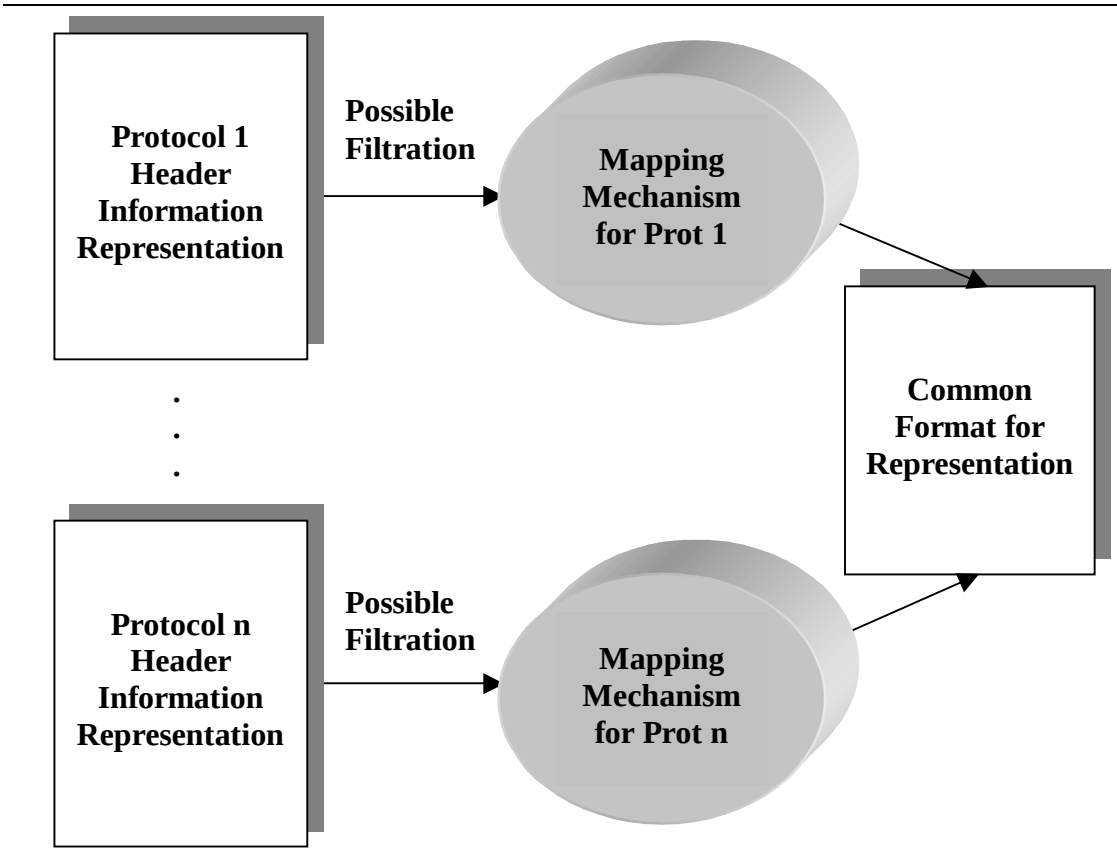


Figure 12: Desirable protocol header mapping architecture

The somewhat different focus of the requirements considered the representation and manipulation of data on a more general level. The efficiency and optimisation factors were put aside, and it became interesting to investigate the common characteristics of the messages. The aim of this is to somehow gather the essential characteristics of the

messages in a new, common representation. Figure 12 shows the approach for this transformation mechanism for the messages and the data. The desire is to construct a mapping and conversion mechanism for every protocol, enabling a common structure for messages in the other end. The transformation of the data might also be combined with a filtering process, to enable only portions of the data to be transformed, at will.

The expectation of the above mentioned approach is that it may aid in the work to overcome the complexity that comes with the large number of different application protocols, along with their own set of data types and message types. For this to happen, a set of data types and message types has to be defined, that will cover all the necessary characteristics of the presently defined types for all the application protocols.

As discussions continued, the focus was once again altered. Due to the limited time frame of this project, it became obvious to us that it would take too much effort to be able to make an accurate estimation of a possible representative set of universal data types and message types for the protocol integration at EMW. This is because such estimation would require total insight into all the structures of the whole protocol set defined at EMW. That, in turn, is unfeasible not only because of the time constraints, but also because of the secrecy policies at EMW. Therefore, only a part of the requirements will be addressed.

The problems are of the same nature for all data structures in the problem context. This has implied that the need for data integration facilities has been considered not to be of critical importance initially. The reason for this is that all messages are structured in a similar manner, regardless of their type and associated protocol. So, if the problem is approached with example message structures, they could give a representative result to the problems at EMW. The approach will then be to find a suitable way to transform the message data from its specific representation of data types to a more general representation. The new representation would possibly contain only three example types: an integer type, a floating point number type and a holder or container type (similar to an object-oriented class definition).

The new representation of the data is hoped to enable a tree-like structure of the data from the source representation, to build some kind of class hierarchy of the data. As a result of the irregular structure of the data only limited search and filtering is possible, as stated above. For analysis purposes EMW would like this hierarchical way to represent data, to be able to easily access the atomic values from the simulation scenarios.

The creation of a suitable hierarchical representation is expected to enable better search and filtering mechanisms. If data is represented in a suitable fashion, the search and retrieval of information could be done by traversing the hierarchical structure in order to filter out the interesting atomic values. In the end it is hoped that this could lead to more sophisticated analysis possibilities, and also, if the data format is common, searching and filtering can be conducted in the same way for all the different application protocols.

Even though data is to be represented in a new way, the same type of distribution of queries as on the present format is desirable, i.e. one (optional) course-grained filtering mechanism (see Figure 10) and one finer, but with more sophisticated capabilities in the posing of queries, as discussed above. One example query could be "Retrieve all cars from 1992 or newer that are blue and have had a speed of 100 mph at some point between Stockholm and Gothenburg". A wish is to be able to perform navigational search for values in a way similar to the punctual notation when retrieving values in object-oriented representation. One example query could be:

TypeA.subtypeAA.speed > TypeB.speed ?

Since Matlab will use atomic values for analysis, it is critical that the integrity and semantics of data is kept intact throughout the transformation process. For example, if a struct is transformed into an alternative "container" type, it is crucial that it will later be possible to access the information about the struct's origin, i.e. tracability of data is desirable. Additionally, it is also important to preserve the relationships between different parts of the data during transformation. This is because the analysis of the data will get inaccurate otherwise. The risk with this specific to general data transformation process is that some of the important semantic characteristics are lost on the way.

EMW is, based on the discussions above, most interested in finding out to what extent an alternative representation could be used to fulfil its needs in the context of the problematic issues, which the present representation cannot do.

The improvement of the filtration mechanism is also considered important. Finding a new way to filter out detailed information will hopefully eliminate the current problems with searching. This could be done perhaps by replacing the current indexing mechanism.

Even though finding efficient ways for storing data, with respect to data modification issues, might come in handy in a near future, it is not considered a primary problem to address. The main reason for this is that the log-files resulting from the scenarios are created once and are most seldom subjects to change, i.e. modification of the content is very rare. If it is, the modification is done manually. This issue is, however, something that might come in as an integrated part of other, higher prioritised, problems. In such a case, it must certainly be considered.

7 Analysis of XML in the EMW problem context

This section presents the results from an analysis of XML with the requirements and desires of EMW as a basis. The analysis was performed by using the knowledge about the technologies of the XML framework, discussed in chapter 2.

The analysis work was complemented by reviewing research literature issuing XML and its characteristics. This contributed to more information and a broader perspective on the XML technologies, as well as identification of relevant parts of XML.

The XML technologies more focused on web and data exchange issues, e.g. XLink and RDF, did not participate in the analysis. This is because the problems of EMW are not concerned with web technology, and thus such mechanisms are not interesting to evaluate.

The results from the analysis are presented in comparison to the previous relational based solution described in section 2.4. The reason for discussing the earlier solution here is because a possible approach towards XML will be used in the same problem context. This analysis is not aiming to give an exhaustive comparison between the nature of XML and relations. It should rather be seen as a way to make the characteristics of XML in the problem context more concrete. This is most easily done by putting it in contrast to an already considered approach for solving the problems at EMW.

First in this section, an estimation of the possible new and different problems raised with the introduction of XML will be given. Moreover, some advantages that will disappear in case of a decision to abandon the solution described in section 2.4, in favour of XML, are discussed. In conclusion, some new possible advantages of an XML solution are presented.

7.1 New obstacles when adopting XML

EMW is interested in finding out how well another approach for solving the company's problems might work, as compared to the solution that EMW has earlier been outlining. EMW has decided to investigate XML. However, EMW's earlier solution and XML have very different natures and origins. This will cause different situations to come up and, above all, new problems to be encountered, if XML is to be adopted into the company. Some of these problems are described below.

A problem that will arise instantly in a process of transforming the EMW data into XML is the inability to represent binary data in core XML. The core standard permits only text in an XML document (Bray et al., 2000). Allowed characters in the text are whitespaces and the legal characters of Unicode and ISO/IEC 10646. If binary data is to be included, it must either be stored as unparsed data or referenced as an external entity.

As a partial solution to the binary problem, the new XML Schema standard proposes a way to easier integrate binary data. This is done by using a data type called *Base64*, which is used to encode the binary data to suit the textual representation (Biron and Malhotra, 2001). Although this is a step in the right direction to enable the integration of real binary data in XML documents, the data has to be encoded and decoded every time, which could be quite time consuming. Discussions on how to efficiently handle binary data in XML documents have been made by e.g. Rein (1998) and Singh (2001).

One property of XML that will become problematic is its inability to constrain the integrity of the data that it models. Moreover, there are very small possibilities to specify any semantics of XML data. The only semantic meaning to be found is encapsulated within the tags in the document. Not even DTDs can be used for the purpose. They can only help to validate the syntactic structure of an XML document. As we see it, this is bound to be a problem for the employees at EMW if they intend to use XML, since they consider it essential to uphold the integrity and semantic significance of their data.

The use of XML Schema can improve the situation in this area as well. The considerable number of new data types supported, and the ability to define stronger relationships between data, such as inheritance, could improve the constraining of data.

The properties of XML will set new conditions for the data representation at EMW. Two examples are that XML is self-describing, i.e. the schema for the data is replicated within each object of a document, and that XML documents can only contain text. Also, the replicated schemas imply inefficiencies such as time costs for retrieval of information (Deutsch et al., 1999b).

Storing the binary data as text has implications for EMW. While the binary form of e.g. a positive integer between 0 and 255 is stored as one byte, its textual correspondence could be stored in one to three bytes. This is because a text string takes up space of one byte per represented character. Binary numbers do not necessarily take up more space when represented as text. Storing binary data as text could both be more and less efficient, depending on what is to be stored. However, the complex binary data structures at EMW mostly consist of integers and floating numbers. These numbers often demand a high degree of precision in their representation, and the more precision a number requires, the less efficient is storing it as text. In addition to this, the tag definitions for all the data in XML will also take up space of one byte per character. Here the specific representation of empty tags could be used to decrease the damage. Nonetheless, it is not unreasonable to estimate an increase of 10-15 times in space needed. The space needed to store Gigabytes, or even Terabytes, of complex binary data structures at EMW as text, will then take enormous proportions.

XML is still pretty unknown to the personnel at EMW. This means that an approach towards integrating XML into the company will involve a learning period. This will involve the usual problems when learning a new paradigm, but also problems related directly to XML. The origin of XML is quite different from that of the data at EMW. Even worse, the debated coverage of the XML standard makes it harder to determine what is important to master in XML.

7.2 Limitations of XML

Apart from describing the new problems related to XML, it is important to identify the advantages that will be lost if an XML approach is taken by EMW, as opposed to the relational based solution, which was presented in section 2.4.

It is argued in Kappel et al. (1999) that XML is often referred to simply as a "data format", not having an appropriate data model. This is a big disadvantage compared to EMW's earlier approach based on relational technology, which had its foundation in structured and well-established data model. This was probably an unconsidered, but nonetheless a welcome, strength of the earlier solution.

Another disappearing advantage with XML is that it has very few established standards. As noted earlier, the XML coverage is very hard to identify, much of the proposed XML framework is still ongoing work. Once again, the solution that EMW outlined before has an advantage, due to the mature and widely spread standard of SQL92 and relational technology (Kappel et al., 1999).

As a result of the limited standardisation of XML techniques, there is today a lack in experience in the use of XML. This is of course because XML still is a very fresh standard. The consequence of this is that there currently exist very few applications and tools designed specifically for XML (Mecca et al., 1999).

The seamless querying facilities and well known syntax of the SQL92 query language will also be lost with XML. It can be recalled that there does not even exist an agreed query standard for XML data. In addition, there are some general drawbacks when querying data of semistructured kind (Suciu, 1998). First, queries are hard to formulate efficiently. This is because even a simple path expression may require the whole document to be traversed. Second, the queries are harder to formulate, because the structure of the document is not always fully known.

Although no query language standard exists for XML, the query like facilities of XSL could be used in the problem context, since the querying demands of EMW were not very complex. In our opinion, XSL's intimate relationship with XML, along with its technical maturity and widespread implementations, makes it the most promising candidate for use by EMW in the querying context.

One of the biggest advantages that will disappear with XML is the possibility to specify constraints on the data, and to define the semantics of it. Although the relationship modelling facilities of the earlier solution was criticised by EMW, it would still provide far better facilities for constraining data. Since it was based on SQL92 it could be used to define several types of constraints, including domain and key constraints. Furthermore, it would make it possible to specify entity and referential integrity, along with various types of dependencies, e.g. functional dependencies (Elmasri and Navathe, 2000).

XML considerably lacks the restriction facilities enumerated above. There does however exist some ways to restrict the data in an XML document. The most common way is of course to use DTDs, but they are not well suited to define semantics, at least not in comparison with database schemas.

Another feature in XML that could potentially be used in this context is the ID/IDREF mechanism. However, even if the similarities between the ID/IDREF in XML and the primary key/foreign key in relational database systems are easy to identify, there are also big differences. The constraining power of the ID/IDREF mechanism is much more primitive than that of relational keys (Fan and Siméon, 2000). This is e.g. because IDREF attributes are untyped.

The domain constraints in relations specify allowed values within certain ranges for attributes. These must conform to common elementary data types, such as integers and floating numbers. There are a large number of supported domains in SQL92. Hence, the solution in section 2.4 would give EMW much freedom if used in this context. In contrast, the domains supported in XML are very few. For element types, only one single atomic domain can be specified (PCDATA), while attributes can belong to three different

domains. This loss of support for domains if EMW would adopt XML instead of the solution EMW outlined earlier is bound to lead to problems in a mapping process.

The semantic shortages of XML could to a great extent be traced to the narrow modelling possibilities, i.e. only text allowed in a document. The XML Schema standard might in this situation help to simplify the work (Fallside, 2001). With its support for over 40 different data types, it will enable seamless mapping of much more data than is possible with only the core XML standard.

Path constraints can be specified, to in some way constrain the integrity of XML data. Work has been done with propositions on how to enforce more integrity on XML data, e.g. Fan and Siméon (2000), and on how to add more semantics into XML, e.g. Psaila and Crespi-Reghizzi (1999).

A disadvantageous characteristic of XML is its loose notion of schemas. The subject for schema definitions in XML is the DTD. However, as opposed to solutions based on the relational model, schemas are not mandatory in XML (Mecca et al., 1999). If they do exist, they are embedded in the data itself. Thus, there does not exist any explicit independent schema constraining the document. This is another reason for the semantic problems of XML.

An additional shortage of XML is that it does not have a specific mechanism for meta data storage, which the EMW project members also considered to be the case with their analysed solution. This actually shows a misinterpretation made by EMW. Meta data would most probably be possible to specify for the relational based solution, by utilising e.g. a data dictionary (Elmasri and Navathe, 2000).

7.3 Possible benefits of using XML

When having identified the disappearing advantages of using XML, we consider it to be equally important to analyse the new advantages with XML in the EMW problem context.

XML documents are by nature hierarchically structured. This is an advantage in the EMW context, since a conceivable mapping of the EMW data will be simplified. The desideratum of EMW is that its data can keep the structural dependencies in the target representation. The hierarchical representation of data in XML makes it much more structurally convenient to use for EMW than the flat representation in EMW's relational solution. Also, XML DTDs are ordered. This is in contrast to relational tables, where no order is assumed (Mecca et al., 1999).

The way EMW data is structured can in many ways be mapped to object-oriented concepts. This provides for another of XML's advantages in the context of the data at EMW. Suciu (1998) suggests that XML documents should be seen as object-oriented structures. The element types defined in a DTD would then be the class definitions, the elements in the XML document are the instances or objects of the defined classes, while the attributes associated with certain element types are variables of the classes. In the same way, ID attributes could be seen as object identifiers. We believe that this resemblance between the two different ways of representation should simplify the structural mapping. However, the resemblance in structure does not give any guarantees of straightforward mappings of relationships in the data.

One of XML's most known advantages is extensibility, i.e. an object based system has the ability to define new abstract data types from existing types. Similarly, the complete absence of predefined tags in XML makes it unrestricted and extensible. The possibility to specify own tag sets, to suit the specific application at hand, brings great freedom to the user. The extensibility makes it easy to add new data to the existing system, without any complex redesign of schemas.

Another commonly mentioned advantage of XML is its flexibility. The flexible nature of XML is achieved by encoding the schema within the data (Deutsch et al., 1999b). This provides XML with the ability to represent data of many different kinds. Moreover, it enables XML to handle irregularities and variations in the structure of data. This property of XML gives the opportunity to represent the more complex types of data in EMW files,

e.g. unions and sequences, thereby avoiding the modelling problems that the EMW project group had with SQL92.

The flexibility of XML also allows it to be used efficiently for data integration. Since it is not necessary that the structure be known in advance, the data from new sources can be loaded and integrated with the existing data immediately (Deutsch et al., 1999b). Thus, many kinds of data from different origin can be represented in the same way, a feature very attractive to EMW.

In addition to the earlier enumerated advantages provided by XML's flexibility, there is the improvement in schema maintenance. The replicated schema makes even big changes in the structure of the data very easy to handle (Deutsch et al., 1999b). This is an advantage over schema management in EMW's relational based solution, where schemas would have to be defined a priori, and therefore less easy to maintain. Consequently, big changes might require a complex schema redesign effort (Connolly et al., 1996, p. 685).

Erdmann and Studer (2001) argue that the extensibility and flexibility functionality is the biggest advantage of XML. However, this is also its biggest handicap. As mentioned above, the reason that enables XML to model practically any kind of data is its possibility to integrate the schema with the data. At the same time, this feature prohibits XML from specifying sophisticated semantic constraints on data.

XML gives the opportunity of navigational querying on data. This is typically in line with the requirements of EMW. In contrast, the SQL92 query language, which was to be utilised in the solution described in section 2.4, can only be used for associative access to data. This was probably an unexpected drawback of SQL92 for EMW.

Last but not least, a new advantage that will be acquired with XML is portability (Psaila and Crespi-Reghizzi, 1999). XML is both a platform and language independent format. This makes XML powerful and further extends its flexibility. Also, it might contribute to the spread of XML in several areas.

8 Conclusions

It has been shown that XML has characteristics that make it both better and worse equipped than EMW's solution based on relational technology in the EMW problem context. The most obvious advantage of XML is that its extensible and flexible characteristics make it much better equipped than the earlier outlined approach to represent the complex kinds of data at EMW, e.g. unions. Furthermore, XML's features make it suitable to be used for data integration at EMW. Other strengths of XML are schema maintenance and portability.

The main reason that XML is worse equipped than EMW's earlier analysed solution to represent and manipulate the EMW data, is because XML once and for all was built primarily for textual processing. No matter how the XML technologies are adapted to specific environments, it will not be possible to overlook the fact that XML has its foundation in the document processing community. This has put a stamp on XML, e.g. weak integrity and semantics, which will implicate a big effort if XML is to be used for something it was not originally designed to handle. This is actually the case in the EMW context.

XML Schema could be used as a complement to the core XML standard techniques, to enforce more integrity and semantics into the data. However, this will breed new problems to EMW. Instead of only having to learn to master the core XML standard, the employees at EMW need to familiarise themselves with a whole new schema language. Also important to notice here, is that in Whitlock (2001) it is revealed that even W3C insiders have the opinion that XML Schema is very complex to handle. John Clark, the technical lead for the XSLT standard, calls XML Schema "too complicated" as it is today to be practically usable. There are no sophisticated tools to simplify the use of XML Schema, since it has only been a standard recommendation since May 2001. The conclusion has to be that in the semantics and data integrity domain, XML is a very poor alternative.

When it comes to querying, XML has features for navigational search through data, which is a desire at EMW. A detriment to the use of an XML query language is that no standard exists. However, XSL includes facilities that might at least be used for simple query processing on XML data.

Although XSL seems sufficient to use for the relatively limited query complexity of EMW, it is still under development and therefore subject to change.

The criticism against XML above does not implicate that EMW necessarily should settle for the relational based solution. As has been described, there are areas in the EMW problem context in which XML could be considered more suitable to use than EMW's former approach.

Since XML is still in its infancy, there are practically no applications or tools to aid in a possible approach to use it for the needs of EMW. It is also hard to estimate how this will change in the nearest future. There is still much speculation on what XML can really be used for. It is therefore today impossible to foresee where and how the debate around XML will finally settle. Widom (1999) has thoughts around XML as a storage format, while Deutsch et al. (1999a) have visions about XML replacing SQL as query standard. However, the most points at XML being used primarily as a data interchange format between businesses, in addition to the markup functionality it was originally designed for. This is also the opinion of the majority of the database research community, e.g. Psaila and Crespi-Reghizzi (1999). An implication will be that the development of tools of the kind attractive to EMW will probably be very sparse.

8.1 Future work

We consider it to be interesting in the EMW problem context to investigate the storage of XML documents. Furthermore, it would be relevant with an evaluation of different storage strategies and approaches to identify efficient ways for storing XML data. In this context,

related work of e.g. Tian et al. (2001) might be useful. Data compression work like the XMILL project (Liefke and Suciu, 2000) is also of interest.

Another possibility for future work is to evaluate the usable extent of other approaches than EMW's relational based solution and, first and foremost, XML. It would be very interesting if analyses of similar approaches based on e.g. object-oriented, object-relational and/or hierarchical database technology were performed. These analyses could then present comparable results in the EMW problem context. They could e.g. show the usable extent of newer query language standards, i.e. OQL and SQL99.

It would also be interesting to know more about how commercial database solutions might be used to solve the problems of EMW, as opposed to the internal database approach discussed in this report.

Since it was completely disregarded in this work, optimisation techniques for e.g. query evaluation and information retrieval is a research area of interest to EMW.

As an answer to this work, a survey and evaluation of commercial XML tools and techniques for the purpose of EMW is attractive. Such a survey could give a description of present and future directions in the market.

References

Abadi M. (1999) *"Protection in Programming-Language Translations"*, Secure Internet Programming 1999, pp 19-34.

Abiteboul S. (1999) *"On views and XML"*, Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'99), pp 1-9.

Abiteboul S., Cluet S., Milo T., Mogilevsky P., Siméon J., Zohar S. (1999) *"Tools for Data Translation and Integration"*, Bulletin of the Computer Society Technical Committee on Data Engineering, Vol 22, No 1, pp 3-8.

Adler S, Berglund A., Caruso J., Deach S., Grosso P., Gutentag E., Milowski A., Parnell S., Richman J., Zilles S. (2000), *"Extensible Stylesheet Language (XSL)*, Version 1.0, W3C Candidate Recommendation, 21 November, available at: http://www.w3.org/TR/xsl.

Atzeni P., Torlone R. (1995) *"Schema Translation between Heterogeneous Data Models in a Lattice Framework"*, Proceedings of the Sixth IFIP TC-2 Working Conference on Data Semantics (DS-6), pp 345-364.

Bernstein P. A., Bergstraesser T. (1999) *"Meta-Data Support for Data Transformations Using Microsoft Repository"*, Bulletin of the Computer Society Technical Committee on Data Engineering, Vol 22, No 1, pp 9-14.

Biron P.V., Malhotra A. (2001) *"XML Schema Part 2: Data types"*, W3C Recommendation, available at http://www.w3.org/TR/xmlschema-2/.

Bonifati A., Ceri S. (2000) *"Comparative Analysis of Five XML Query Languages"*, SIGMOD Record Vol 29, No 1, pp 68-79.

Bosak J. (1997) *"XML, Java and the future of the web"*, Sun Microsystems, published in World Wide Web Journal, 1997.

Bosworth A. (1998) *"Position Paper for the W3C query language Workshop – A proposal for an XSL Query Language"*, Proceedings of the Query Languages Workshop, World Wide Web Consortium, available at http://www.w3.org/TandS/QL/QL98/pp/microsoft-extensions.html.

Bray T., Hollander D., Layman A. (1999) *"Namespaces in XML"*, W3C Recommendation, available at http://www.w3.org/TR/REC-xml-names/.

Bray T., Paoli J., Sperberg-McQueen C M., Maler E. (2000) *"Extensible Markup Language 1.0"*, Second Edition, W3C Recommendation, available at http://www.w3.org/REC-xml.

Buneman P. (1997) *"Semistructured data"*, Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS'97), pp 117-121.

Chamberlin D., Clark J., Florescu D., Robie J., Siméon J., Stefanescu M. (2001a) *"XQuery 1.0: An XML Query Language"*, W3C Working Draft, available at http://www.w3.org/TR/xquery/.

Chamberlin D., Fankhauser P., Marchiori M., Robie J. (2001b) *"XML Query Requirements"*, W3C Working Draft, available at http://www.w3.org/TR/xmlquery-req.

Chawathe S. S. (1999) *"Describing and Manipulating XML Data"*, Bulletin of the Computer Society Technical Committee on Data Engineering, Vol 22, No 3, pp 3-9.

Chinwala M., Malhotra R., Miller J. A. (2001) *"Progress Towards Standards for XML Databases"*, Proceedings of the 39th Annual ACM Southeast Conference (ACMSE'01), pp 277-284.

Clark J. (1999) *"XSL Transformations"*, Version 1.0, W3C Recommendation, available at http://www.w3.org/TR/xslt.

Clark J. and DeRose S. (1999) *"XML Path Language (XPath)"*, Version 1.0, W3C Recommendation, available at http://www.w3.org/TR/xpath.

Connolly T., Begg C., Strachan A. (1996) *"Database Systems: A Practical Approach to Design, Implementation and Management"*, Addison-Wesley Publishing Company, England.

DeRose S., Maler E., Janiel Jr R. (2001a) *"XML Pointer Language (XPointer)"*, Version 1.0, W3C Last Call Working Draft, available at http://www.w3.org/TR/xptr.

DeRose S., Maler E., Orchard D. (2001b) *"XML Linking Language (XLink)"*, Version 1.0, W3C Recommendation, available at http://www.w3.org/TR/2001/REC-xlink-20010627/.

Deutsch A., Fernandez M., Florescu D., Levy A., Suciu D. (1998) *"XML-QL: A Query Language for XML"*, Proceedings of the Query Languages Workshop, World Wide Web Consortium, available at http://www.w3.org/TR/1998/NOTE-xml-ql-19980819/.

Deutsch A., Fernandez M., Florescu D., Levy A., Maier D., Suciu D. (1999a) *"Querying XML Data"*, Bulletin of the Computer Society Technical Committee on Data Engineering, Vol 22, No 3, pp 10-18.

Deutsch A., Fernandez M., Suciu D. (1999b) *"Storing Semistructured Data with STORED"*, Proceedings ACM SIGMOD International Conference on Management of Data (SIGMOD 1999), pp 431-442.

Dimitrov M. (2000) *"XML Standards for Ontology Exchange"*, in Proceedings of "OntoLex 2000: Ontologies and Lexical Knowledge Bases", Sozopol, Bulgaria.

Dubinko M., Dietl J., Merrit R., Ragget D., Raman T. V., Welsh L. B. (2001) *"XForms 1.0"*, W3C Working Draft, available at http://www.w3.org/TR/xforms/.

Elmasri R., Navathe S. B. (2000) *"Fundamentals of Database Systems"*, Third Edition, Addison-Wesley Publishing Company, USA.

Erdmann M., Studer R. (2001) *"How to Structure and Access XML Documents With Onthologies"*, Data & Knowledge Engineering (DKE), Vol 36, No 3, pp 317-335.

Fallside D. C. (2001) *"XML Schema Part 0: Primer"*, W3C Recommendation, available at http://www.w3.org/TR/xmlschema-0.

Fan W., Siméon J. (2000) *"Integrity Constraints for XML"*, Proceedings of the 19th ACM Symposium on Principles of Database Systems (PODS'00), pp 23-34.

Fernandez M. and Suciu D. (1998) *"A Query Language for XML"*, Technical Report, Proceedings of the Query Languages Workshop (QL'98), World Wide Web Consortium, available at http://www.w3.org/TandS/QL/QL98/pp/att-position-paper.html.

Goldman R., McHugh J., Widom J. (1999) *"From Semistructured Data to XML: Migrating the Lore Data Model and Query Language"*, ACM SIGMOD Workshop on The Web and Databases (WebDB'99), pp 25-30, Informal Proceedings, INRIA.

Haas L. M., Miller R. J., Niswonger B., Tork Roth M., Schwarz P. M., Wimmers E. L. (1999) *"Transforming Heterogeneous Data with Database Middleware: Beyond Integration"*, Bulletin of the Computer Society Technical Committee on Data Engineering, Vol 22, No 1, pp 31-36.

Halsall F. (1996) *"Data Communications, Computer Networks and Open Systems"*, Fourth Edition, Addison-Wesley Publishing Company, USA.

Hart K. W., Searls D. B., Overton G. C. (1994) *"SORTEZ: A Relational Translator for NCBI's ASN.1 Database"*, Computer Applications in the Biosciences, Vol 10, No 4, pp 369-378.

ISO (1986) ISO 8879:1986(E), *"Information processing – Text and Office Systems – Standard Generalized Markup Language (SGML)"*, First Edition, Geneva, Switzerland.

James J., Honegger C. (1996) *"Mil-Std-1553: A tutorial"*, VMEbus systems, USA.

Jansz K. (1998) *"Intelligent processing, storage and visualization of dictionary information"*, Sydney University Language Technology Research laboratorY (SULTRY), University of Sydney, Australia.

Kappel G., Kapsammer E., Retschitzegger W. (2000) *"X-Ray – Towards Integrating XML and Relational Database Systems"*, 19th International Conference on Conceptual Modeling (ER 2000), pp 339-353.

Lassila O., Swick R. R. (1999) *"Resource Description Framework (RDF) Model and Syntax Specification"*, W3C Recommendation, available at http://www.w3.org/-TR/1999/REC-rdf-syntax-19990222/.

Liefke H., Suciu D. (2000) *"XMILL: An Efficient Compressor for XML Data"*, Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD 2000), pp 153-164.

Maier D. (1998) *"Database Desiderata for an XML Query Language"*, Technical Report, Proceedings of the Query Languages Workshop (QL'98), World Wide Web Consortium, available at http://www.w3.org/TandS/QL/QL98/pp/maier.html.

McHugh J., Abiteboul S., Goldman R., Quass D. and Widom J. (1997) *"Lore: A database management system for semistructured data"*, SIGMOD Record Vol 26, No 3, pp 54-66.

McHugh J. and Widom J. (1999) *"Query Optimization for XML"*, Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), pp 315-326.

Mecca G., Merialdo P., Atzeni P. (1999) *"Araneus in the Era of XML"*, Bulletin of the Computer Society Technical Committee on Data Enginering, Vol 22, No 3, pp 19-26.

Muench S., Scardina M., Fernandez M. (2001) *"XPath Requirements"*, Version 2.0, W3C Working Draft, available at http://www.w3.org/TR/xpath20req.

OMG (2001) *"CORBA/IIOP Specification"*, OMG Specification, version 2.4.2, available at http://www.omg.org/cgi-bin/doc?formal/01-02-01.

Papakonstantinou Y., Garcia-Molina H., Ullman J. (1996) *"MedMaker: A Mediation System Based on Declarative Specifications"*, Proceedings of the Twelfth International Conference on Data Engineering (ICDE), pp 132-141.

Pemberton S., Altheim M., Austin D., Boumphrey F., Burger J., Donoho, A W., Dooley S., Hofrichter S., Hoschka P., Ishikawa M., ten Kate W., King P., Klante P., Matsui S., McCarron S., Navarro A., Nies Z., Ragget D., Schmitz P., Schnitzenbaumer S., Stark P., Wilson C., Wugofski T., Zigmond D. (2000) *"XHTML 1.0: The Extensible HyperText Markup Language, A Reformulation of HTML 4 in XML 1.0"*, W3C Recommendation, available at http://www.w3.org/TR/2000/REC-xhtml1-20000126.

Psaila G., Crespi-Reghizzi S. (1999) *"Adding Semantics to XML"*, Proceedings of the Second Workshop on Attribute Grammars and their Applications (WAGA'99), pp 113-132.

Ragget D., Le-Hors A., Jacobs I. (1998) *"HTML 4.0 Specification"*, W3C Recommendation, available at http://www.w3.org/TR/1998/REC-html40-19980424.

Rein L. (1998) "Handling Binary Data in XML Documents", Technical Report, available at http://www.xml.com/pub/a/98/07/binary/binary.html.

Robie J., Lapp J., Schach D. (1998) *"XML Query Language (XQL)"*, Technical Report, Proceedings of the Query Languages Workshop (QL'98), World Wide Web Consortium, available at http://www.w3.org/TandS/QL/QL98/pp/xql.html.

Rosenthal A., Seligman L., Costello R. (1999) *"XML, Databases, and Interoperability"*, Technical Report, MITRE Corporation, San Diego, USA.

Seligman L., Rosenthal A. (2001) *"The impact of XML on Databases and Data Sharing"*, Technical Report, MITRE Corporation, San Diego, USA, published in IEEE Computer, 2001.

Shanmugasundaram J., Tufte K., He G., Zhang C., DeWitt D., Naughton J. (1999) *"Relational Databases for Querying XML Documents: Limitations and Opportunities"*, Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99), pp 302-314.

Singh D. (2001) *"XML and Binary Data"*, Technical Report, available at http://www.perfectxml.com/articles/xml/binary.asp.

Suciu D. (1998) *"Semistructured Data and XML"*, The 5th International Conference of Foundations of Data Organization (FODO'98).

Thompson H.S., Beech D., Maloney M., Mendelsohn N. (2001) *"XML Schema Part 1: Structures"*, W3C Recommendation, available at http://www.w3.org/TR/xmlschema-1.

Tian F., DeWitt D., Chen J., Zhang C. (2001) *"The Design and Performance Evaluation of Alternative XML Storage Strategies"*, Department of Computer Science, University of Wisconsin, Madison, USA, submitted for publication.

W3C XSL Working Group (1998) *"The Query Language Position Paper of the XSL Working Group"*, Technical Report, Proceedings of the Query Languages Workshop (QL'98), World Wide Web Consortium, available at http://www.w3.org/TandS/QL/QL98/pp/xsl-wg-position.html.

Watson A. (2000) *"CORBA and XML; conflict or cooperation"*, Object Management Group Whitepaper, available at http://www.omg.org/news/whitepapers/watsonwp.htm.

Whitlock N. W. (2001) *"XML Schema becomes W3C Recommendation"*, Technical Report, available at http://www-106.ibm.com/developerworks/xml/library/x-schrec.html?dwzone=xml.

Widom J. (1999) *"Data Management for XML: Research Directions"*, Bulletin of the Computer Society Technical Committee on Data Enginering, Vol 22, No 3, pp 44-52.

Wüthrich, M. (1998) *"An Authoring System for Multidimensional Documents"*, Computer Science Theory Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland.