Open Formats and Interfaces for Exchange of Public Information

by

Michael Eric Menk

Master Thesis in
Information and Communication Technology

Agder University College

Grimstad, May 2003

Summary

Electronic reporting is an important part of all national eGovernment initiatives. XML4DR is a standard based on XML, proposed by a consortium funded by the European Commission. An application used for reporting from municipalities to central government has been developed by Comfact AB.

A platform independent application will allow each municipality to freely chose tools such as browser, text processor and operating system.

A platform independent application will rely on either open standards or a set of vendor specific pieces of code for the deviation form the standard.

In this thesis I have evaluated the possibility of using W3C DOM to implement a reporting system using the results from the IQML project. Since DOM is still in a rapid development, Internet browsers have just started to develop the support for W3C DOM. Also the application developed by Comfact have some vendor specific code, because the needed methods were not available in any browser at the time of development.

To determine the feasibility of using DOM for the reporting system I have compared the required functionality of the reporting application to the functionality defined in DOM and the standard compliance of some Browsers. To confirm the results I also developed a “hello world” application, that only followed standard.

I have found out that the standards that are defined by standardization organizations cover the required functionality for implementing the hello world application. Unfortunately, currently none of the browsers supports all the functionality needed to implement the hello world according to the DOM standard. Currently, Mozilla is the browser with the best standard conformance. Only the XML uploading of a file to the server is missing.

Preface

This thesis was written as a part of the Master of Science in ICT degree at Faculty of Engineering and Science, Agder University Collage. The work has been carried out between January 2003 and May 2003.

I would like to thank my supervisor, Mikael Snaprud at Agder University Collage for valuable help. I would also like to thank Anders Tornkvist at Comfact AB, and Herman Robak at Opera Software. I would also like to give tanks to people on Usenet, Jonas Koch Bentzen, Martin Honnen, Stig Nygaar and Heikki Toivonen.

Grimstad, May 2003

Michael Eric Menk

Tables of Content

Summary....2
Preface....3
Tables of Content....4
Table of Illustrations....6
List of Tables....7
1 Introduction....8
1.1 Background....8
1.2 The goal of the project....8
1.3 Literature....9
2 Relevant Standards....11
2.1 HTML....11
2.2 xHTML/XML....11
2.3 CSS....11
2.4 DOM....11
2.5 UN/EDIFACT....12
2.6 IQML....12
2.7 XFORMS....12
2.8 CSS, XSLT and DOM....13
2.8.1 CSS....13
2.8.2 XSLT....13
2.8.3 DOM....15
2.8.4 Why keep XSLT....17
3 DOM....19
3.1 Basics....19
3.2 Levels and modules....20
3.3 DOM support....20
4 KOSTRA....23
4.1 FormFlow....23
5 The IQML Project....24
6 Demonstrator....26
6.1 Specification of demonstrator....26
6.2 Implementation of demonstrator....26
6.2.1 Selection of primary browser....26
6.2.2 Compliance of browsers....27
7 Discussion....28
7.1 Respect of Standards....28

7.1.1 HTML....28
7.1.2 Open Source....30
7.2 Browser Compliance....30
7.2.1 Implementation of standards....31
7.2.2 Program Safely....32
7.3 The different browsers....32
7.3.1 Developer tools....33
7.4 Web developers....34
7.5 Saving XML file to disk....34
8 Conclusion....35
Software Reference....36
Abbreviations....37
Reference....38
Appendix A - A DOM example, loading an external XML file....40
Appendix B - MSIE code to appendix A....42
Appendix C - xml-save func. for App. A....43
Appendix D - Output from App. A and B....44
Appendix E - hasFeature() results....45
Appendix F - hello world source code....46
Appendix G - Software used in this thesis....57

Table of Illustrations

Illustration 1 Overview of e-government interactions [1].....9
Illustration 2 HTML 4.01 browser on a b/w 12MHz computer.....11
Illustration 3 XSLT basic layout.....14
Illustration 4 DOM Usermap.....15
Illustration 5 DOM interface.....16
Illustration 6 A simple client side DOM example. Upper image is before the script is run.....18
Illustration 7 A DOM HTML-tree.....19
Illustration 8 Digital reporting, simplified.....25
Illustration 9 The Norwegian Public roadmap. By the Statens vegves and Statens Kartverk, requires Microsoft product. The dialog box translation: 'The application is only supported by Microsoft IE 5.0 or above'.....29
Illustration 10 A DOM page in DOM and non-DOM browsers.....31
Illustration 11 The JavaScripts console in Mozilla, Opera and MSIE.....33
Graph 1 Browsers DOM compliance claims.....21

List of Tables

List of DOM modules and levels.....20

1 Introduction

1.1 Background

Development of eGovernment solutions have led to a number of different solutions. Different sectors have made their own solutions to fit their needs. Different entities of the same type (eg. counties) have made their own systems. The different systems are often performing the same task. However, they are not necessarily interoperable.

Making and maintaining a large customized system is complex and costly. Sometimes, the systems have been bundled into even larger systems. When a part of such a system is outdated, the reporting system also have to be upgraded. This in turn, leads to a slower shift from manual reporting to electronic. Manual reporting from local government to central government bodies has been an error prone time consuming process involving manual key punching both for producing the report and for entering the data at the receiving end.

One of the problems with the systems having closed communication protocols and formats is that if this is utilized by the government, all citizens and organization needs to use the same vendor owned format and buy the products from the company providing them. Some further important reasons for using open standards are described by the Sincere Choice campaign[2] open standards are the key to a market with free competition. The individual users, businesses, and governments are free to choose their own policies, regardless of the choices of others.

There have been attempts of creating open standards before. The UN/EDIFACT, a United Nation standard for intercommunications between governments, companies and organization. UN/EDIFACT never became a widely used standard for intercommunication. Mainly because the standard was so complex, that proprietary communication formats were often used instead.

The IQML (Intelligent Questionnaire Markup Language) project was started, to make a new standard, based on XML[3]. This enables programs to use the vast number of tools that are made for the XML format. An other advantage with XML, is that the format is also easier to read and understand for humans. One of the technologies that can be used with this new standard is W3C's DOM. DOM enables a programmer to manipulate tag based documents. By utilizing DOM, a web browser can generate reports, using the XML file from the IQML project. However, as with most standards, this will only work, when all parties follows the DOM standard.

The problem is that a standard is just a peace of paper, stating how the standard works. If no one implements the standard, it's just that, a peace of paper.

1.2 The goal of the project

The goal of the thesis is to investigate the possibility of using W3C/DOM for

implementation of a platform independent reporting system for public information. To test for conformity to standard needed to implement the protocol that was derived from the IQML project.

The system XML file is mainly to be used in Government-to-Government (G2G) communication. G2G can for instance be communication between governmental organizations to a central statistical organization. Although, it possible to use it in the future for other communication, like between government and citizen (G2C) (Illustration 1).

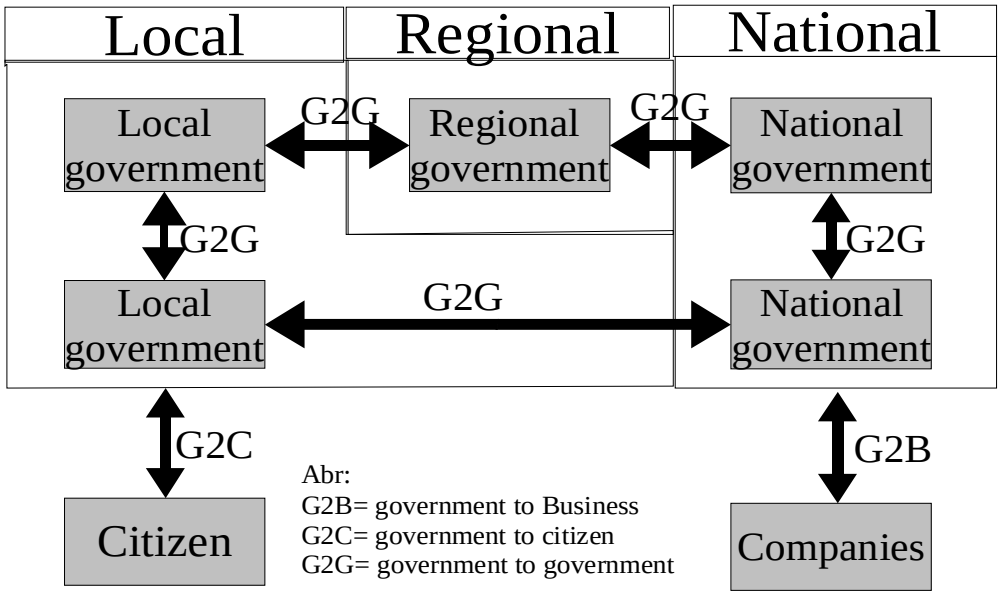


Illustration 1 Overview of e-government interactions [1]

In short, the main activities in the projects have been as follows;

- Determine the functionality needed for a reporting system.
- Define what parts of different standards that may be used, to implement such a system.
- Test support of browsers of these standards.
- Implement a small Hello world application, using these standards.

1.3 Literature

The main sources of literature, is W3C's standards Recommendations and “Specification of RDRMES XML Version 0.1” [4] from the IQML project. W3C is the organization that defines the web standards, like HTML[5], XML[3] and DOM[6].

One of the challenges, when surveying for sources of DOM information, was that many of the books praise DOM. However, many of the books give the impression that the

example is DOM, and gives no warning, that the code is vendor specific.

The only books I was able to find, that followed the standard, and commented when using non standard code, was books from O'Rielly. The two books used the most, were “JavaScript, The Definitive Guide. 4 Edition”, by David Flanagan (January 2002)[7] and “Dynamic Html, The Definitive Reference, 2. Edition”, by Danny Goodman (September 2002)[8]. The non standard code, was always a fall back, if the standard was not supported. All tags and methods listed were referred to the standard they are defined in, and proprietary method and tags were listed under which browser they are supported by.

Usenet have also been a great source for finding the relevant reference, where is seems to be a growing support for following standards.

2 Relevant Standards

2.1 HTML

Hypertext Markup Language (HTML), is a tag based language for representing text. Unlike other SGML DTDs, HTML tags are more directly linked to their visual appearance. With tags for paragraph, headings and links. While other SGML files, like DOCBOOK, have tags that represent the type of data. The file is then later rendered by the author to e.g. HTML.

Illustration 2 HTML 4.01 browser on a b/w 12MHz computer

HTML is the most used document format on the web. The document is stable. And if made correct, a document will look nice on a large screen, like a PC, and a small b/w hand held computers.

Cascading Style Sheet (CSS)[9] is recommended by W3C to use for the formatting of the documents, while the HTML file contains only the content. By keeping the formatting and content separated, it's easier for the web developer to develop degradable pages, as illustrated in Illustration 2.

2.2 xHTML/XML

xHTML is the 'HTML DTD' of XML instead of SGML. The names of the tags are the same, except they are case sensitive, and have to be written with lower case. What's special with xHTML, is that it's possible to change the XML to e.g. MathML[10] inside the document.

It's also a possibility of using pure XML in a browser. The tags have then no meaning, and everything have to be defined in a CSS file or a XSL file.

2.3 CSS

Cascading Style Sheet[9] (CSS) is a separate, or in line document for web browsers. The style sheet can define font size, color &c. CSS can also separate between different media, such as print, screen, presentation, handheld (Illustration 2)

2.4 DOM

Document Object Model (DOM) is a W3C recommendation. DOM is a set of methods for accessing tag based documents like XML and SGML. The core of the DOM, is a set of methods, that applies no matter what kind of document you are accessing. XML based documents, like xHTML, SMIL or SGML documents, like HTML or DOCBOOK.

The DOM was started for the purpose of standardizing the methods name that where used in browser. Where the methods for accessing different HTML elements had

different names. There are in addition to the core part of the DOM, some extensions for different formats, like HTML, CSS &c.

DOM can be used to replace current systems in Norway, for digital reporting from Citizens to government (C2G). One of the electronic forms that can be replaced with DOM and XML, is the governmental return form for travel expenses. This electronic report is currently using a system based on Microsoft Word. The forms come as self-executables.

2.5 UN/EDIFACT

The current standard for intercommunication is 'United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport' (UN/EDIFACT). UN/EDIFACT is published as an ISO standard (ISO 9735). The UN/EDIFACT consists of a set of UN/EDIFACT Standard Messages (UNSM). There are a number of subject expert groups, that develop and maintain the over 200 different messages. Some example of the messages are data messages (PRODAT, D1 - Materials Management), an invoice message (INVOIC, D6 – Purchasing), a payment order message (PAYORD , D6 Finance) and many more[11].

The complaints about UN/EDIFACT, is that with it's 2700 pages, it's so large that no single program is able to implement the whole standard within reasonable resources. The UN/EDIFACT is an ASCII based protocol, which makes it even more difficult to implement. In response to this, there have been started an open source project, to make a UN/EDIFACT <-> XML converter. This open source project is called EDIFACT-XML[12].

2.6 IQML

The goal for the project was to make an XML based system for digital reporting between governmental agencies in Europe. The system was suppose to be flexible enough to use in all the agency in Europe, and be able to change easily, as the logging parameters of the agency changes over time.

The advantage with XML, is that it's is developed a lot of tools for working with XML. which makes it a lot easier to work with than with ASCII based protocols and file formats. One of the tools avalieble is DOM compatible web servers' modules and browsers.

One of the important elements with the formats that came out of IQML, is that it will be easier to implement than the current standard, UN/EDIFACT. The fact that it's easier to implement, makes it more affordable to smaller organizations, and in the governmental case, save the taxpayers money. The project is an EU funded project.

2.7 XFORMS

XFORMS[13] became a W3C recommendation on November 2002, so it is a relative new standard. The purpose of XFORMS is to separate Xforms model, instance data, and

user interface. XFORMS is not a free-standing document type. To be useful, it must be link up with something else, like XHTML or SVG.

In this thesis, I have not looked in to this brand new standard.

2.8 CSS, XSLT and DOM

2.8.1 CSS

Cascading Style Sheets[9] (CSS) is a file for defining layout for HTML and XML files. With HTML 4 Strict, no layout tags are allowed int the HTML file. All for the layout must be in a style sheet. All style based fonts, like <font> and some restriction on layout based attributes are not allowed in HTML 4. This makes a clean HTML, and completely separates content from layout. The HTML elements already have some style built into the different tags. For instance that <H1> is bold font, and is larger than the <H2> font. With style sheets, its possible to change the default attribute of the tags. CSS can also be used in a XML file. But then there is no predefined attribute on the different tags.

2.8.2 XSLT

XSLT (eXtensible Stylesheet Language for Transformations) is a style sheet for XML. In web-browsers, it's also possible to use CSS to apply style to an XML file, but XSLT is a more powerful. XSLT can also be used to generate SMIL,SVG, VRML, (x)HTML, PDF and more from a XML file. The XML file contains the content, and the XSL contains the layout and formating (HTML, SMIL &c. tags). By separate content, the XML file can easily be used for human reading, with the use of the XML-file together with the XSL file. The same XML file, can eg. Be imported into an other database (Se Illustration 3). It’s also possible to render the XML and the XSL on a web server, and send a HTML document to the client. This server side rendering, will make it possible for non XML/XSLT compliant browsers to render the document.

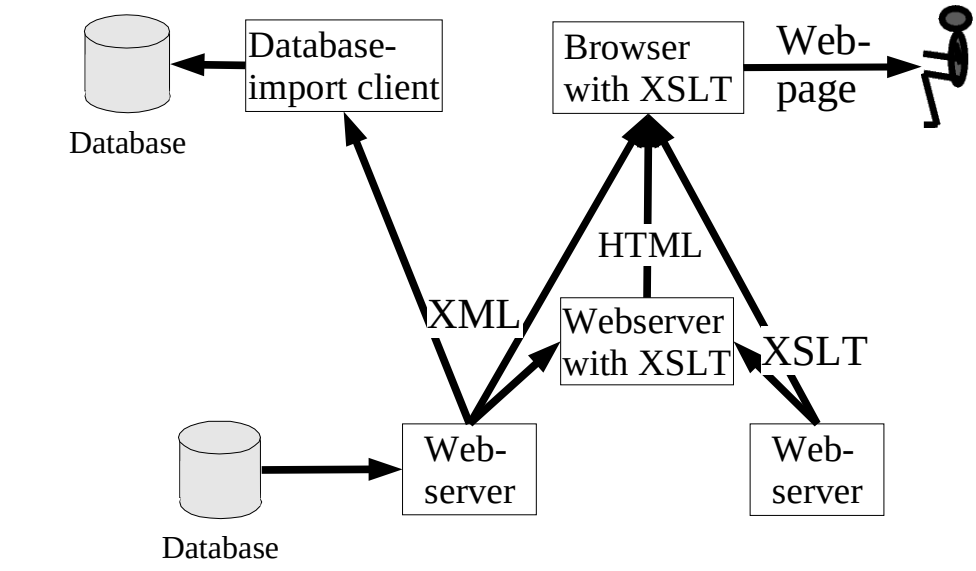


Illustration 3 XSLT basic layout

The following files, is a hello-world example[14] in use of XSLT :

XML-file:

```
<?xml-stylesheet
     type="text/xsl" href="style.xsl"?>
 <greeting>
  Hello, World!
 </greeting>
```

Stylesheet:

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

 <xsl:template match="/">
  <xsl:apply-templates select="greeting"/>
 </xsl:template>

 <xsl:template match="greeting">
  <html>
   <body>
    <hr />
    <h1>
     <xsl:value-of select="."/>
    </h1>
    <hr />
```

```
    </body>
   </html>
  </xsl:template>
 </xsl:stylesheet>
```

2.8.3 DOM

The DOM user map (Illustration 4)starts to be more complex. DOM is not only for XML, but also SGML and CSS. There are subsets of DOM, made especially for XML standards like xHTML, and SGML-DTD's like HTML4.01.

The main difference between XSLT and DOM is that DOM is a interface. A set of commands for a programing language. You get DOM for PHP, pearl, Jscript, javascript, java and more.

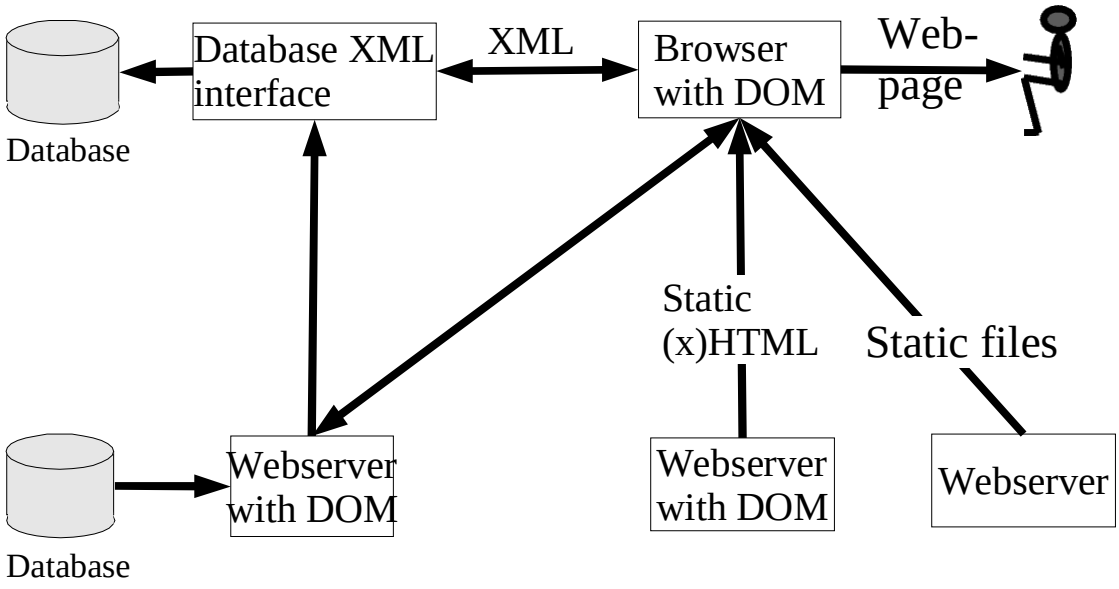


Illustration 4 DOM Usermap

Server side DOM have until now, been the normal way to use DOM. This is because the developer knows what is supported by the implementation, and can avoid unimplemented parts of DOM. A server side system can be used as you would have used PHP or ASP. The clients do not have any knowledge of DOM. With client side DOM, the developers do not know that well what the visiting browser supports. The advantage of client side lower load on the servers is faster response, and less network usage. A manipulation of the document on the server, have to be reloaded in order for the user to see the effect. A server side DOM application would also need session control, to be able to keep track of the status of each client using the DOM application.

The major difference in use of XSLT and DOM, is the ability to manipulate the document. You can change the DOM tree. In other words, edit the content loaded document.

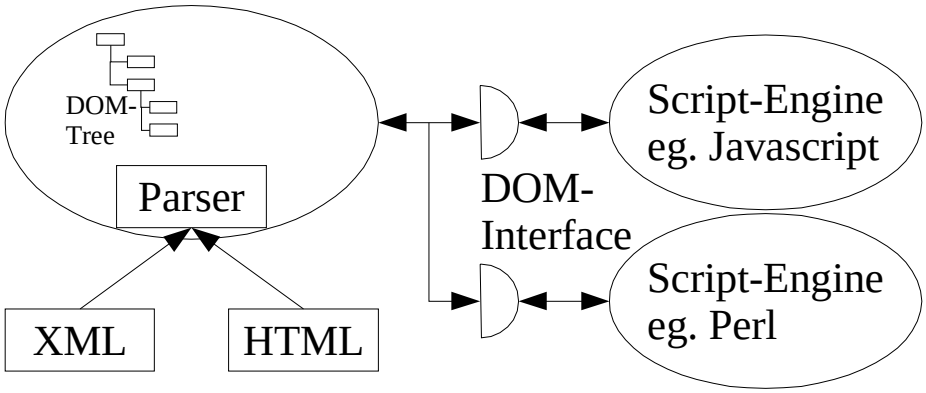


Illustration 5 DOM interface

The DOM methods are called trough interfaces that are implemented in the script engine(Illustration 5).

A small client side example of DOM HTML.

```
// Javascript file - javafile.js
function setCite(){
    var curr_cite = document.getElementsByTagName("q");
    if (curr_cite.length == 0 ){
        my_addLi ("There ar no qoutes in this document");
    }
    for ( var i = 0 ; i < curr_cite.length ; i++ ){
        my_addLi ( curr_cite[i].innerHTML + " (" +
          curr_cite[i].cite + ")" ,"quotelist" );
    }
}
function my_addLi( text , id ){
    var cite_p = document.getElementById ( id );
    var tmp_li = cite_p.appendChild(
          document.createElement("LI"));
    tmp_li.appendChild ( document.createTextNode( text  ));
}
function my_clear_ul (){
    document.getElementById("quotelist").innerHTML="";
}//Javascript file
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="HTML Tidy for Linux/x86 (vers 1st March
2003), see www.w3.org" name="generator" />
<title>cite</title>
<script type="text/javascript;version=1.5"
src=”javafile.js”></script>
```

```
</head>
<body>
<p>To quot someone: <q cite="http://www.example.com/">To be
or not to be, thats the question of life.</q></p>
<p><q cite="_This_is_the_cite_var.">The Q tag is used
here.</q></p>
<p><a onclick='setCite();'>List Quotes</a>
<a onclick='my_clear_ul();'>Clear Quotes</a></p>
<div id="qpar">
<ul id="quotelist"></ul></div>
</body>
</html>
```

The example is a xHTML file. The file contains two quotes. When “List Quotes is pressed, the script adds the source of the quotes, and the quotes to the element with ID=”quotelist” (Se: Illustration 6 at page 18).

The list elements are added, by getting the values from the <q> tags, with DOM. New elements are created with DOM, in the wanted element. This element is accessed by ID.

2.8.4 Why keep XSLT

Some people believe that we don't need XSLT when we have DOM. You can see these questions around on different Usenet and other web based BBS.

The arguments against XSLT, usually comes from DOM programmers, that don't see any reason for learning something else, when it can be done with ECMA script and DOM. There are currently companies that block ECMA script for security reason. With the use of DOM to generate for instance a HTML document from the slashdot.xml (http://slashdot.org/slashdot.xml) would not work, because the script files would have been deleted by the firewall. To render the XML document, you also would have to have ECMA script engine. With XSL you could render the document with the native language of the browser e.g. C++. DOM is not language independent. An ECMA file will only work in other ECMA compatible script engine. Even though Phyton supports DOM will it not be able to manipulate the file. Since XSLT is language independent. The same file can be used in Python and in the web browsers.

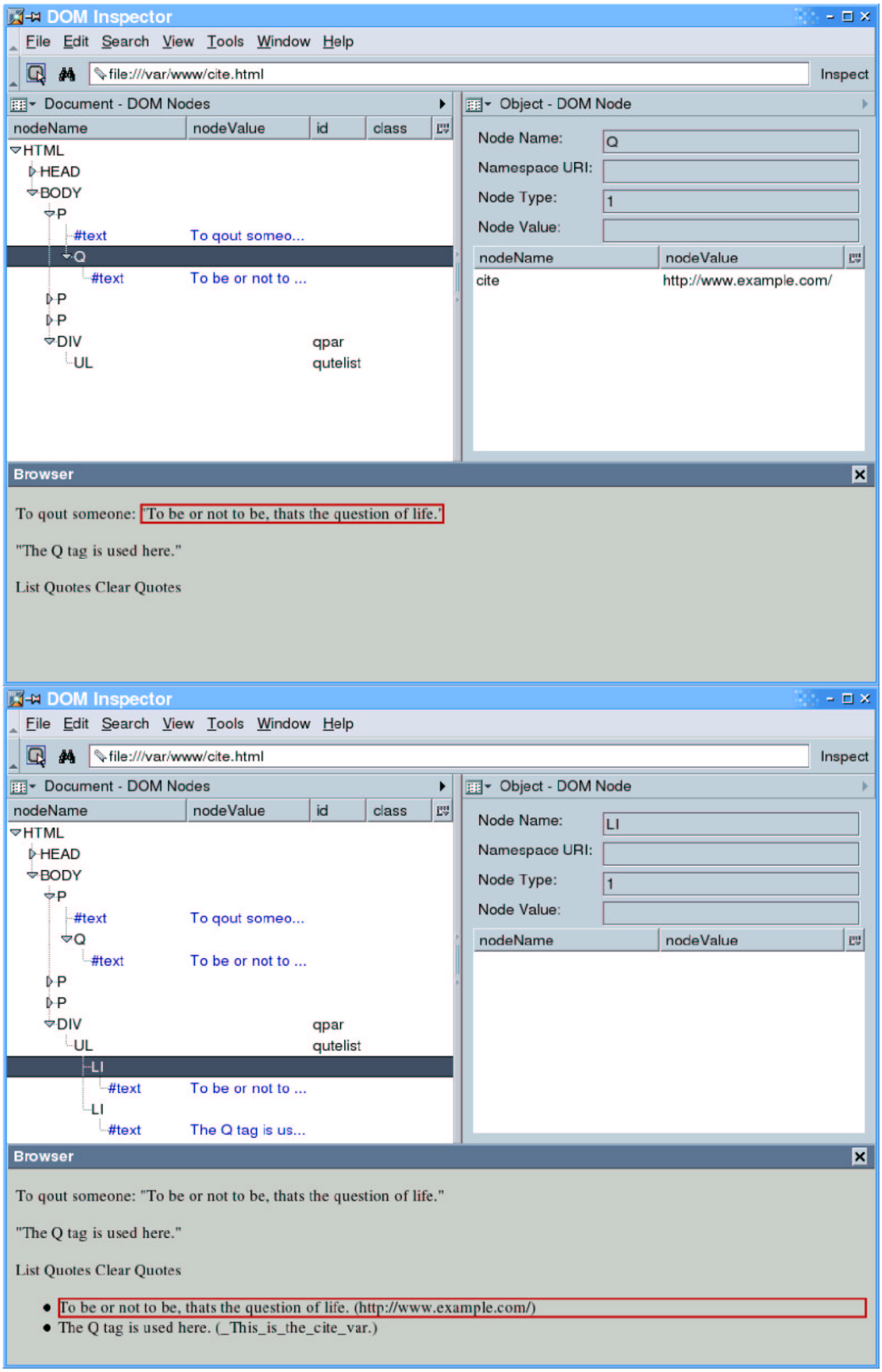


Illustration 6 A simple client side DOM example. Upper image is before the script is run.

3 DOM

3.1 Basics

DOM is not a file format, but a way do access and manipulate tagged documents like XML and HTML (a subset of SGML). The DOM implementation takes eg. a parsed HTML document and presents trough the DOM interface. The interface is in Object management Group's (OMG) Interface Definition Language (IDL). The interface allows a program to access the document. The document is represented as a tree. When this three is manipulated, the tree structure ensures that the document's tags are consistent.

Below is a short example of how DOM works. The tree is what you assess through the DOM interface, not the document i self. To simply the example, some HTML elements are left out. The HTML document in the example is therefor not a valid HTML document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN">
<HTML>
 <H1>Title</H1>
 <p>Blah....</p>
</HTML>
```

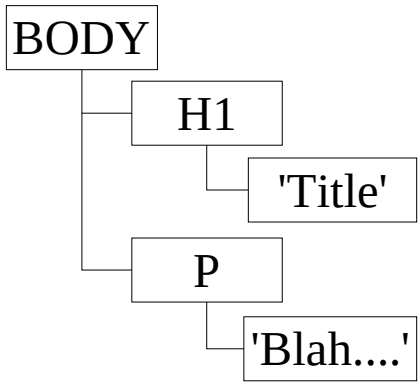


Illustration 7 A DOM HTML-tree

If we would want do delete from 'le' to 'Bl'. By editing the HTML document, we could by mistake get an invalid document like this.

```
<HTML>
 <H1>Titah....</p>
</HTML>
```

By editing the tree we would be sure to get.

```
<HTML>
 <H1>Tit</H1>
```

```
 <P>ah....</P>
</HTML>
```

With the cutout being:

```
<HTML>
 <H1>le</H1>
 <P>Bl</P>
</HTML>
```

3.2 Levels and modules

The DOM is divided into Levels and Modules. The levels can easiest be understood as version numbers. Each version are divided into several modules. The Core modules, is a general module, that can manipulate all tag-based files. The HTML modules are a special module, to manipulate HTML files. There are methods that are specific to the HTML standard. One of these method are getElementById(string). The reason this method is not in the Core module, but is in the HTML module; is that there is no rule that states that no more than one element in XML can have the same value of the attribute id. In HTML only one element can have the same id value.

When it comes to implementation and compliance, there are certain dependencies. A DOM implementation can not claim support for a module in a certain level, without having support for the Core in the same level. The same applies to the same module, in the previous level. In other words, if your software supports DOM Level 2 HTML, then it also supports DOM Level 1 HTML, DOM Level 1 Core and DOM Level 2 Core.

Module	*Levels*	*Module*	*Levels*	*Module*	*Levels*
Core	1 2 3	**CSS2**	2	**HTML Events**	2 3
XML	1 2 3	**Events**	2 3	**Range**	2
HTML	1 2 3	**User Interface Events**	2 3	**Traversal**	2
Views	2	**Mouse Events**	2 3	**Load and Save**	3
Style Sheets	2	**Text Events**	3	**Abstract Schemas Editing**	3
CSS	2	**Mutation Events**	2 3	**XPath**	3

Table 1 List of DOM modules and levels

3.3 DOM support

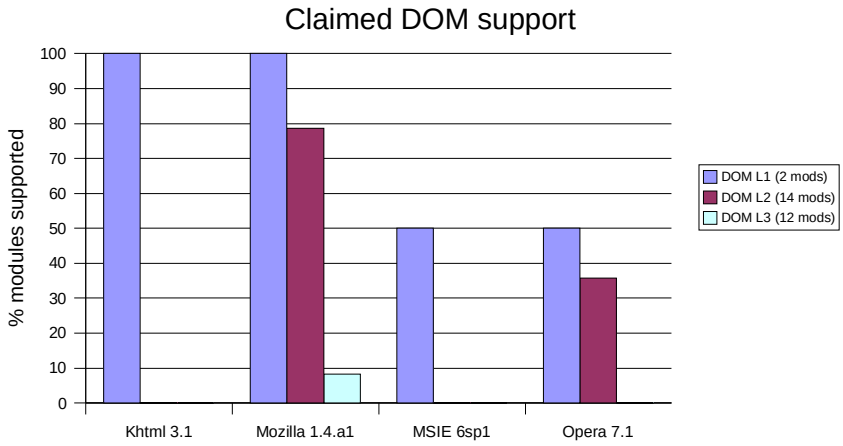
There have mainly been to types of developers; the one type codes the safe code, keeping to standards, ensuring that the page works in as many types of browsers as possible. The other type, likes to be on the leading edge, and uses fancy feature. Some of these developers do not care if the page only works in their own browser. In the past, Netscape was the browser of choice, but now the tables have turned, and now Microsoft Internet Explorer have taken over the role for developers that only develop for one

browser. Many of these sites have a test on the name of the user agent, resulting in messages like the one you get on “Visveg” (Illustration 9). On some sites, you aren't allowed in to the site, even if you supports the features that they use.

To combat this problem, all DOM implementations have to have a function, that will return true/false if they support a feature 100% or not.

```
if ( document.implementation
    && document.implementation.hasFeature
    && document.implementation.hasFeature('CORE', '2.0'))
    /*Testing if the hasFeature exists, before calling it */
{
  //menu by ID. (the static menu)
  menuTable=document.getElementById('menu');
  //runs kode, to make dynamic menu.
  generateMenu();
}//else -> Your browser don't support DOM2core
```

If we had tested on the user agents, the browsers (by May 2003) that do not support DOM Level 2 Core, would have skiped the dynamic menu, even after newer version of the browser came out, with DOM Level 2 Core support. With this code Opera, KHTML and MSIE will automatically run the generateMenu(); function when their done implementing the DOM module.



Graph 1 Browsers DOM compliance claims

Unfortunately, the hasFeature() method is what the vendor claims that they support. We can assume that when they return false on a hasFeature() method, it is reasonable to believe, that that’s the case. If the browser returns true, there is a possibility that this is the truth. MSIE 6 claims trough its hasFeature() method that it supports DOM Level 1 Core and HTML, this is actually not the case.

The most likely problems a developer can encounter, is the node-type constraint, which is defined by the Node interface[15]. It can be noted that in MSIE 6sp1 on W3C “What do you browser claim to support-site” (http://www.w3.org/2003/02/06-dom-support.html) only DOM Level 1 XML returns true. The other browser may also have some elements in a module that they don't support, even if they claim support through the hasFeature() method. In response to this, W3C have made a program, where you can

check a browser, to see if the browser really supports the module that hasFeature() method claims that it does. They have currently only realest a test for DOM Level 1, and is currently working on one for DOM Level 2. The “DOM conformance Test Suites” (http://www.w3.org/DOM/Test/) is under the GPL.

4 KOSTRA

If the standards are to complex to implement, only the larger companies have the economics to develop and maintain a system over time. This can be one of the reasons who only 2% of American SMEs uses an EDI system, while 98% of large enterprises use EDI[11].

KOSTRA (Kommune-Stat-Rapportering = “municipality statistical reporting”) is the Norwegian reporting system in to SSB (Statistics Norway).

The system currently uses UN/EDIFACT. The system uses the RDRMES (Raw Data Reporting MESsage) type. The client program is a closed source program, that each entity have to by. The program the governmental organization have to have, is FormFlow 99. The communication is carried by SMTP (Simple Mail Transfer Protocol)

SSB is one of the governmental organization, who have participated in the development of the new system for digital reporting (XML4DR).

4.1 FormFlow

FormFlow is a program that is being used by several governments. FormFlow is a closed source program. The program uses UN/EDIFACT REMDES message. REMDES stands for Raw Data Reporting Message. Since the date is raw, and not standardize, other applications using the UN/EDIFACT REMDES message in not necessarily compatible to another program also using REMDES. In that sense, from an interoperability standpoint, REMDES can be viewed a separate protocol. There are no free filler software available for use with FromFlows REMDES messages[16].

The US federal government uses FormFlow for their reporting to the federal government. Where you have to use FormFlow 2.1 or newer [16]. KOSTRA used FormFlow earlier, before they went for the XML4DR. At KOSTRA you needed FormFlow 99 or newer.

The disadvantage with FormFlow, is that each agency must have a FormFlow license. An other disadvantage of FormFlow, that it only is available in Microsoft executables.

5 The IQML Project

According to XML-edifact[17], the UN/EDIFACT standard is 2700 page programmers' nightmare. The standard is so extensive, that its difficult to implement. The UN/EDIFACT standard has tried to make one standard, which covers all commercial/public needs, regardless off branch or origin. This has rendered the standard large and complex. The UN/EDIFACT has over 200 messages covering versus domains.

The IQML project aimed to make a standard that was easier to implement, and had better multilingual support. The XML format from the project is called XML4DR. The syntax of XML4DR is XML. XML4DR uses the UN/EDIFACT Raw Data Reporting Message (RDRMES) message type. RDRMES is a message type[18]. where there are no constraints on the underlining messages, in contrast to the other message types, where everything is strictly defined. The advantage with XML, in contrast to an ASCII protocol or file, is easy too read by humans and thereby implementing it is easier, even if the definition is large.

The IQML project was a part of the European Union Fifth framework Research and development program. The IQML project started January 2000.

IQML XML design goals.

- *Allow for different types of raw data reporting*
    - *from manual data entry using a questionnaire for data entry*
    - *to automatic extraction of data without visualization*
- *Allow for validations using*
    - *data formats and types*
    - *defined rules and constraints*
    - *code lists that can be large*
- *Allow for dynamic guidance when data is visualized*
    - *Using HTML constructions and conventions*
    - *that can be controlled by constraints*
- *Allow calculations in order to*
    - *reduce data input when providing data manually*
    - *use values in validations and dynamic guidance*

(source: User Guide for IQML Version 2.0 [19])

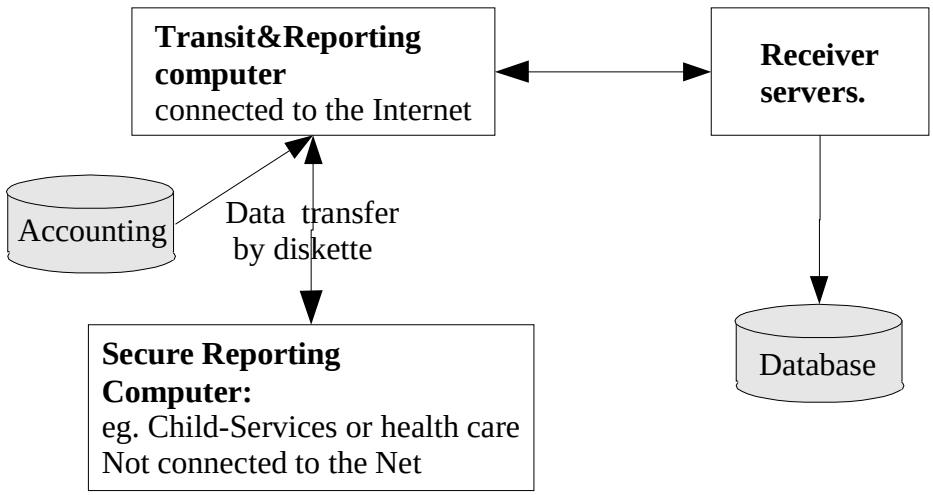


Illustration 8 Digital reporting, simplified

The XML file can be used in more than one way (Illustration 4). For instance between to Database servers, from server to computer with human interaction, and to the final server. In the system used by KOSTRA the needed files are downloaded from the server. The client then fills out the form. The filling of the form, can also implement semi automatic filling, like accounting data from the accounting database server or program. For computer not connected to the net, for instance the Child-Services, the file can be transferred by diskette.

XML4DR may in the future be used for Citizen-to-Government interactions. The registration of the water meter is one of the reports that could have been filled out with XML4DR. The citizens can access the web, and fill out an online form. Companies would maybe want to have water meters that automatically filled out the XML file, and sent it in. Thereby saving labor cost.

There are currently systems in use, for registering the water meter. One of these municipalities is Lillesand, Norway. In 2002 in Lillesand you have to have Microsoft Internet Explorer to register the water meter. By using open standards, other browsers can be used to register the water meter.

6 Demonstrator

The demonstrator is simply a Hallo world program. The basic properties of the program, is to interpret an IQML file. The program then makes a web page for the browser.

The demonstrator is to follow the DOM specification, with no vendor specific extension.

The demonstrators implementation goals were:

- Getting the needed files from a server.
- Generate a form from the XML file, and display it in the web-browsers.
  The form includes:
    - Text fields
    - Drop down (multiple choice fields)
    - Check box
    - Text to the fields listed above, with multilingual support.
- HTTP put back to the server, of the manipulated XML file.

6.1 Specification of demonstrator

Drop down, and checkbooks was not implemented due to time limitations. Implementing these functions, is a trivial task, not needing added functionality from the browsers.

There were no browsers that supported the DOM standard for uploading a file. In absence of DOM compliant browsers, vendor specific code for the HTTP-put was utilized. The vendor specific code, was ActiveX for windows and xmlHttp for Mozilla. The xmlHttp have the same methods as ActiveX's XMLHTTP. Mozilla's xmlHttp may be discontinued when DOM Level 3 Load and Save is implemented.

A DOM browser not supporting this vendor specific extension, will still be able to run this application, but not upload the XML file to the server. The modified XML file is viewable on the web page, underneath the form.

6.2 Implementation of demonstrator

6.2.1 Selection of primary browser

When using DOM to implement IQML, there are basically to ways of dealing with the interface. One is using XML the DOM only to manipulate the data file. A XSL file is used to render the interface. The other option is to have the script engine, read the XML file, and manipulate on a other file like a HTML or xHTML file. Thereby using only DOM and ECMA with no XSL. Due to the fact that Opera 7.1 and KHTML 3.1 do not yet have a XSLT engine. Opera have on their page on what they support[20], a link to

“Formatting Objects considered harmful”[21] by Håkon W. Lie, CTO of Opera Software, and a member of the W3C's Advisory Board, stating that XSLT could not be a client side standard because of accessibility issues.

With two browsers not supporting XSLT, and one of them being against it for accessibility issues. I chose to generate a HTML interface, a manipulating it with DOM.

The primary browser that I chose to program and test against was Mozilla. There are several reasons for this. Mozilla is on the leading edge. The browser also has very nice developing tools, like the JavaScript Console, and the DOMInspector. It's also possible to get the browser to multiple platforms, including the one of my choice, GNU/Linux. The major factor was that the Mozilla browser, was the only browser of the four major browsers, that in January 2003 was 100% done with the DOM Level 1. The implementation of DOM Level 2 was 78% of the modules finished.

Later Opera catched up with 100% DOM Level 1 support, and 35% DOM Level 2 support. And KHTML with 100% DOM Level 1. MSIE claims support only for one of two of the DOM Level 1 (Se Appendix E and Graph 1).

6.2.2 Compliance of browsers.

Even if a browser does not claim to support a module, some of the functions can have been implemented. One of the examples is KHTML, which do not support one DOM module, but here are features that it supports. A browser can not claim support, until all the mandatory features of a W3C standard is implemented. In the meantime, there can be some of the features that work, even if they don't claim support. During the course of the thesis, KHTML started to claim support for DOM Level 1.

There is an other way to find the support of the browsers, this by going to their web sites. At Konquors web site, with a sentence saying that they support DOM. The same was the case for Microsoft Internet Explorer. In both cases, the hasFeature() method and the web pages came with conflicting reports of support.

Opera have a very clean list. Not only do they say what they support, but what they do not support. One of the detailed lists over standards that it partly support, is DOM. Where you can find table, of what they support and what they don’t support.

Mozilla bares the look of a developers site, which is what it is. The information here can be a little much, if you just want to se if a method is supported. There are bug reports, and discussions on what part to implement next. Here users can vote for what to implement next

7 Discussion

7.1 Respect of Standards

The software's compliance with the standards differs from vendor to vendor. There are some that believe, that the level of compliance is more a result of culture, than of know-how. Companies that have disregarded standards in the past, are more likely to do so in the future, with new standards.

When standards are being followed, the users don't notice, and therefore don't think about it. One example of standards that works is the GSM system. You can make a call with your Nokia telephone, trough an Ericsson GSM network, to an old analog phone. We don't think about this, because it works.

7.1.1 HTML

HTML is an example of a standard gone wrong. Although there are other example, many people see the result of the lack of standard compliance on the web.

The Software

There are advanced functions in the newest HTML standard (4.01), coupled with CSS[9]. Two of the major HTML-render engines (Opera and Mozilla) supports these features, but the dominant vendor (MSIE) don't support these features, like hovering items. They have their own tags to do the same features. Hovering is a part of W3C CSS2[9]. There are W3C standards that are from 1996 (PNG[22]), that is implemented in Opera, Mozilla, but not MSIE and KHTML. KTHML level of compliance is better than MSIE[23].

The browsers are supposed to tolerate errors in the HTML; this is because HTML was originally made to bee written by hand. The tolerance is different from vendor to vendor, even between the different versions.

The Documents

In the beginning, the browser followed the standard. When version 4 of Netscape and Internet Explorer, the different vendors tried to gain marked share by adding their own tags and have their own editor, which makes the same invalid tags. Two vendors competed against each other by diminishing the HTML-standard. The earlier dominant vendor, Netscape, lost. This vendor open sourced the core of it browser, which returned to a standard compliant browser.

There are very few that follows the standards for the document they claim to use. When it comes to HTML documents, many of the documents don't even have a DOCTYPE-tag. This tag says to the application what type of SGML type and version the document is. When there is no DOCTYPE, the application can not be curtain that it's a HTML 2.0, HTML 4.1 or DOCBOOK 4.2 document.

The percentage of fully compliant pages on the Internet is under 2%[24]. The number applies to web-pages of different classes. Even public information through the EU, isn't better than the average. Considering that incorrect HTML is a barrier to free competition, and exclude certain users. It do not seam to bee in the best interest of the public to continue publishing public information in such manner.

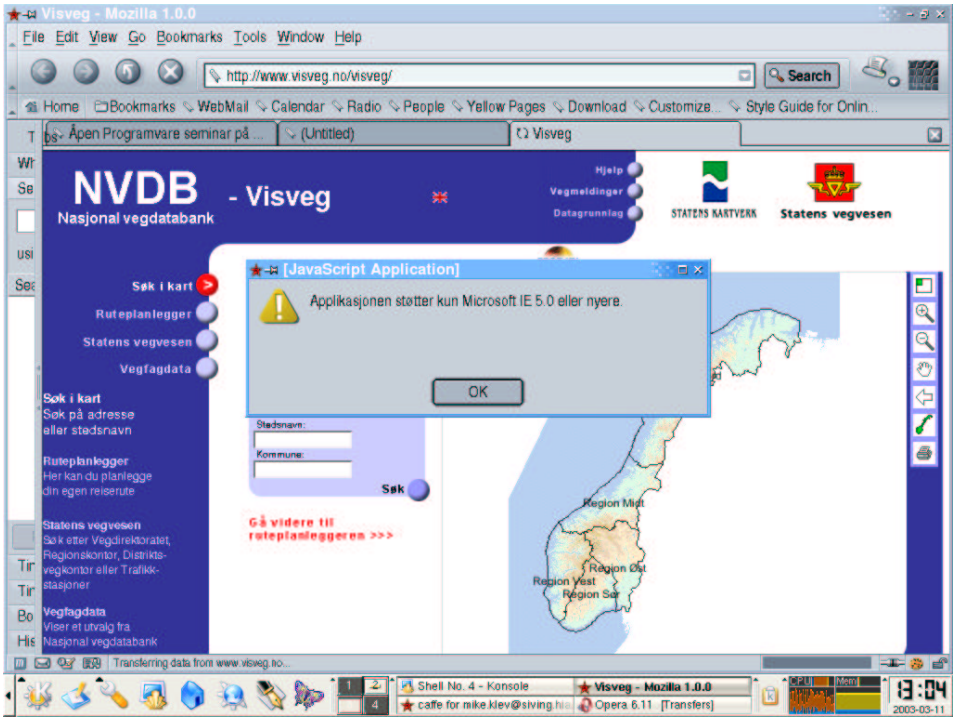


Illustration 9 The Norwegian Public roadmap. By the Statens vegves and Statens Kartverk, requires Microsoft product. The dialog box translation: 'The application is only supported by Microsoft IE 5.0 or above'

If the browsers had not tolerated pages with errors, and the editors did not make up their own tags, the pages with errors would not work, and maintained pages would then have been fixed.

There are some vendors and developers who refer to standard deviation as a necessity for innovations. This is true, for testing innovations, the browser, or the test browser, must be able to read more than the standard. On the other hand, programs that make the file do not need to support these experimental tags. When a person writes in a WYSIWYG HTML editor, or export an office document to HTML, the experiential code should not be used. If the tags are used in productions utilized as default, the tags are no longer experimental, but non-standard code.

According to the SAGA report, from German Federal Ministry of the Interior [1] the W3C xHTML 1.0 standard, is by February 2003 not mature enough or sufficiently proven to use for public information. If W3C documents not are mature enough, what do this say about the maturity of vendor dependent experimental tags and formats? Netscape, now Mozilla, had their share of non standard tags. Most of these tags do not work anymore. The experimental tags and methods have been discontinued in favor of

W3C standards like DOM, HTML, CSS and XSLT.

The Mozilla editor creates valid HTML 4.01 Transitional, while the editors and export functions from Microsoft and Openoffice.org generate HTML documents with non valid tags. Opera and Konqueror do not have editors. The HTML code generated from Macintosh computer, have not been tested, which uses KHTML as the rendering engine in the newest browser.

7.1.2 Open Source

Open source software tends to be more standard compliant. Since the source code is open to the public, locking people to the software by deviating from the standard is less attractive. Its to be noticed that open source do not guarantee that the program follows the standards. OpenOffice.org (Office suite) HTML-export, generate faulty HTML, and the MPEG file from the video editing suite Cinelerra can be exported with invalid frame rate, which makes it unable by all players except their own.

7.2 Browser Compliance

Browsers have in the later years, started to be more standard conformant. This comes after years with a lot of non standard innovations. It is a general consensus among web-designers who vows to the W3C standards, that Opera and Mozilla are the two most standard compliant browses. Unfortunately, Opera have been slow on starting to implement DOM.

Mozilla's compliance of the standards is so good, that some people have problem separating W3C and Mozilla[25]. On the Mozilla developer website, there are a lot of documentation, and non implemented features and standards. On the web site it is possible to vote for which unimplemented standard to implement next.

Opera on the other hand, which is closed source, have a surprising good list over what they support, and what they don't support. The site is very clean, and easy to read. 8 of 16 methods of Element :Node is implemented[26]. On the same page, we can see that Opera do not support the XMLSerializer, which is used in the XML4DR hello world application.

Microsoft Internet explorer, do not have the same straight forward support list as Opera have. On the page stating what MSIE supports, we can read:

> *The Internet Explorer team has put a great deal of effort into providing fast and stable implementations of 100 percent of CSS 1 and 100 percent of DOM level 1 with this [MSIE6] release. With the emergence of other browser versions over the last year supporting these standards, this is clearly a step forward in interoperability of browsers. [27]*

The address to this site was supplied by 'Microsoft Nordic Costumer service'. Unfortunately, the statement above is not true. According to Microsoft own claims, through the document.implementation.hasFeature() method, Microsoft Internet explorer

6sp1 claims that it do not support DOM Level 1 HTML, but the DOM Level 1 XML returns true on the support question.

The most annoying problem, and one a developer have a great chance to encounter, is the lack of support of the constants in the Node interface. This Node is actually inherited by all document-level nodes and elements. By not supporting this constant, all the DOM Modules at all levels are incomplete.

Microsoft is not the only one that claims more support on their web site, than the hasFeature() reports. The claim is that Konqueror do support DOM Level 1 and Level 2 and partial Level 3.

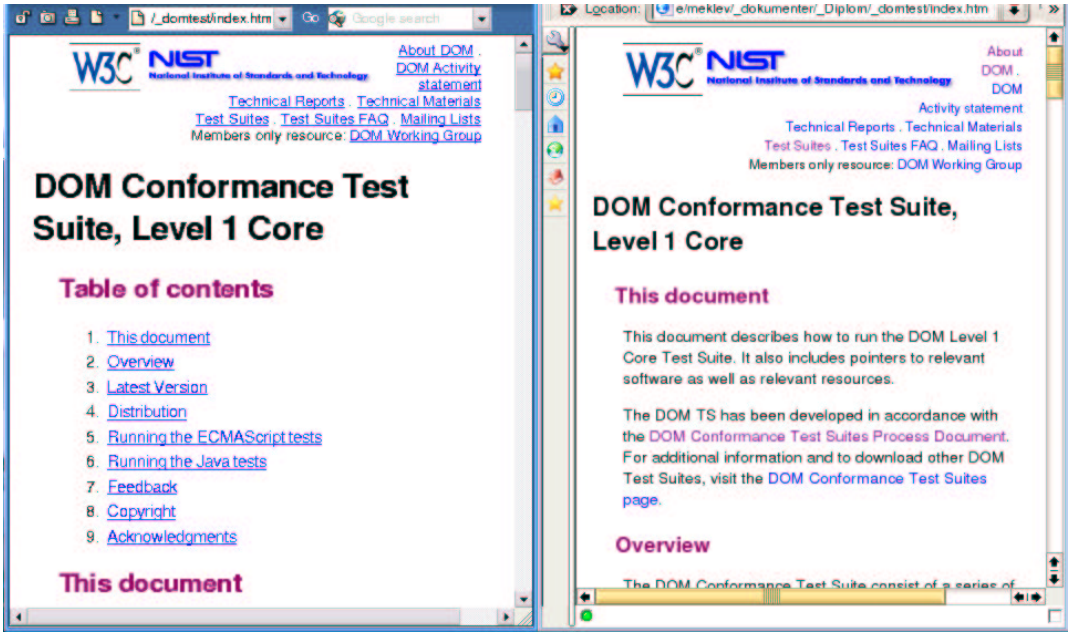


Illustration 10 A DOM page in DOM and non-DOM browsers

7.2.1 Implementation of standards

Standards are just a peace of paper. From this pace of paper from a working application that uses the standard in question takes time. But how long shall we wait. DOM Level 1 came out in 1998, and DOM L2 HTML came out in the spring of 2003. Is it reasonable to assume that a browser supports DOM L1 now? If this is the pace of implementation (4 years), the last browser won't be done with the DOM L3 Load and Save until 2007. Microsoft Internet Explorer is still not done [28] with the PNG[22] format, a W3C standard since 1996. Microsoft promised the users of full PNG support in MSIE4. [23]. The other three browsers support PNG.

> *Microsoft believes very strongly in Internet standards and the standards process, and is committed to implementing appropriate standards when driven by customer demand. [27]*

There is an ongoing petition[23], the petition combined with Microsoft's policy, should bold well for the W3C standard.

7.2.2 Program Safely

There is a saying. “Program Safely”. This is so you do not make a code, which does not work down the road, because the environment has changed. This also applies for the web, even the HTML document. On Web Design Groups web page, they state 4 html errors that in the past, worked in one version of a browser, but not in the next. When writing non standard code, it is less likely that it will work in the next version. In fact, the current code may only work, because of a bug in the browser.

An other advantage of following standards, is that it makes it easier to change system. This in turn create greater competition, which may reduces price[29].

7.3 The different browsers

From a programmers stand point, there was a lot of difference in the four browsers. Konquer was quickly discarded, as it was clear early on that their support for DOM was not good enough. Opera had a nice support list, Mozilla almost had to much on their site. But it is a developer browser, not an end user browser. The Mozilla based browsers do little or none developer info. Microsoft sites were overloaded with more non technical info.

7.3.1 Developer tools

The Mozilla browser is in my opinion the best browser to, with its DOM inspector, and a very nice JavaScript console. Opera's JavaScript console was also good. While Microsoft Ineternet Explorer's JavaScript console, was terrible.

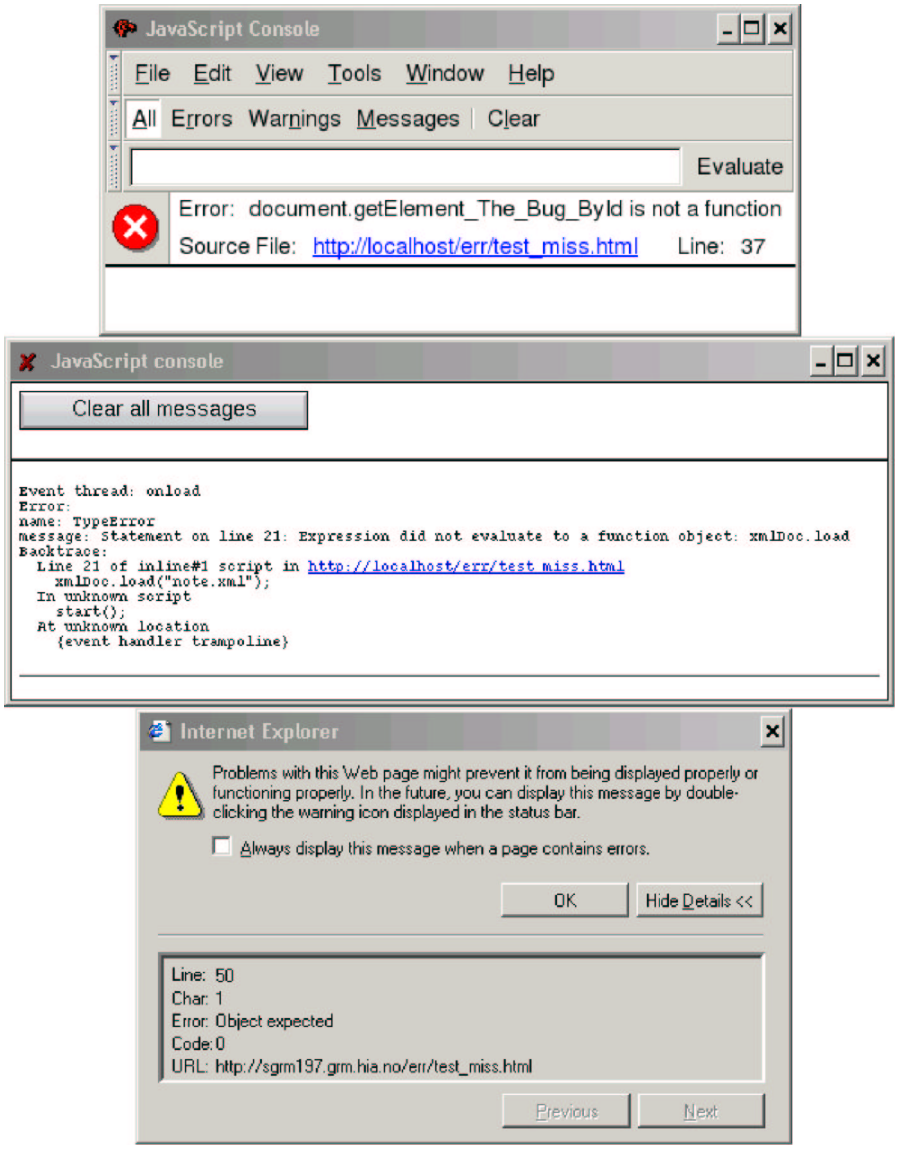


Illustration 11 The JavaScripts console in Mozilla, Opera and MSIE

The consoles can be seen in Illustration 11. The error was generated, by editing line 37 in Appendix A. In Mozilla and Opera, there is plenty of information to find the line number of the error, and to find out why. With Internet Explorer, you get a text, explaining that there is an error in line 50, Line 50 contains "<body bgcolor="" onload="start()"> ".

For a programmer trying to write a W3C DOM application this will lead to frustration.

When the client gives such an inaccurate link to the problem, how can it be expected that the programmer will lay down hours to find out if it's sorting wrong with his code, or the browser. The statement on the vendor web page can not be trusted, as we discuss earlier. I suspect that this is a factor that can raise the cost of making an application.

The hello World application in Appendix F, does not work in Explorer. As the error messages form the MSIE JavaScript console provides little information; I did not further investigate this problem. Finding a problem when the client states that there is an error in your 600 line application, it's like looking for a needle in a hay stack.

7.4 Web developers

Some years ago, people using MSIE complained about pages that did not work. And the pages that did not work, was the one that coded for "Netscape Only". Now, the tables have turned to MSIE. Many of the developers of web pages, do not know about W3C, and have never heard about the standards which is defined by W3C. This is also reflected in the general public, which believe, according to a poll done in the US, asking who created Internet. A large number answered Microsoft Corp or Al Gore (Referd to in a speech[30] by Ralph Naider).

As it is now, many of web developer use vendor specific codes and features. This will hamper the implementation of the standard. The average user do not have the knowledge about standard, and will therefor normally not complain. This is a situation that will benefit the dominant vendor.

Some developers argue something in the line of "over 90% of users uses application X, therefor we don't follow standards, and make vendor dependent code". On the other hand, if that vendor fixes a incomplete standard, 90% of the users now have a standard compliant browser.

7.5 Saving XML file to disk

Mozilla and Opera do not support saving a file to disk from the JavaScript engine. This is because of security reasons. If a script can access the file system, there is a higher probability that a security hole can be found. If such a security hole is being exploited, an attacker can for instance copy out the password file or install a rootkit &c.

There is a way around the block in Mozilla and Opera. If a web server is installed on the same machine, the script can http-put the file to localhost. An other possibility is that a application is specially made to run the XML4DR. There are several script languages that are available for multiple platforms, and supports graphical user interface. One of these is Python. Python have support for DOM, and HTTP protocol and more.

Disabling security features for the purpose of making a reporting system is in my opinion not a good idea.

8 Conclusion

After evaluating the compliance of the standard in the different browsers, it is safe to say, that trusting only the claim on the vendors web site, is not advisable. A clear conflict of statement between the web authors with their support list, and the programmers with the hasFeature() method. When attempting to make standard complaint web an application, then the currently logical choice is to develop towards Mozilla.

All of the vendors indicates on their website, that they support the standards or its processes. Assuming that this is the case, the other browsers will catch up later. In the meantime, exceptions for vendor specific code, can supply temporary support.

The standards that are needed to implement XML4DR on the client side are currently not completely implemented in any of the browsers. Mozilla lack only the save part of DOM Level 3 Load and Save to be able to implement the XML4DR, and following the standards 100%. In the meantime, there is the non standard XMLHttpRequest. The object, that emulates the behavior of Microsoft's ActiveX Object for HTTP transfer, may be removed when DOM Load and Save is implemented.

Opera started with more aggressive drive towards DOM support. During the course of this Thesis, Opera have surpassed KHTML and MSIE on DOM support. The drive towards XSLT support is likely to be slower, because client side XSL is possibility a threat to accessibility[21].

KHTML is the browser with the least support. If an application states what W3C standards it uses and tests on the hasFeature method, then this will encourage the vendors that they support the feature. Microsoft have stated, that they will support the standards, if they are in demand from the costumer[27]. By starting to use the functions that are missing, the demand for the implementation of the lacking standards would increase.

If forced to make browser depended code to get the required functionality, the choice of browser would be Mozilla or Opera, since they are available in multiple platforms. However, the vendor specific code, should always be a fall back from the standard code. An other advantage, from the programmers standpoint, is that Mozilla and Opera have a JavaScript console that gives much more usable information when developing the application.

Software Reference

Name	*URI*
Cinelerra	http://heroinewarrior.com/cinelerra.php3
KHTM	Konqueror: http://www.konqueror.org/ Safari: http://www.apple.com/safari/
Mozilla	http://www.mozilla.org/ Other Mozilla based browsers: http://www.mozilla.org/projects/distros.html
MS Internet Explorer (MSIE)	http://www.microsoft.com/windows/ie/default.asp
Opera	http://www.opera.com/

Abbreviations

CSS	**C**ascading **S**tyle **S**heets
DOM	**D**ocument **O**bject **M**odel
G2B	**G**overnment **T**o **B**usiness
G2C	**G**overnment To **C**itizen
G2G	**G**overnment To **G**overnment
GNU	**G**NU is **N**ot **U**nix
GPL	**G**NU **P**ublic **L**icense
HTML	**H**yper**T**ext **M**arkup **L**anguage
IDL	**I**nterface **D**efinition **L**anguage
IQML	**I**ntelligent **Q**uestionnaire **M**arkup **L**anguage
KOSTRA	**KO**mmune-**ST**at-**RA**pportering
MSIE	**M**icrosoft **I**nternet **E**xplorer
OMG	**O**bject **M**anagement **G**roup
SME	**S**mall and **M**edium **E**nterprise
SMTP	**S**imple **M**ail **T**ransfer **P**rotocol
SSB	**S**tatisisk **S**entral **B**yrå (Eng name: Statistical Norway)
UN/EDIFACT	**U**nited **N**ations Directories for **E**lectronic **D**ata **I**nterchange for **A**dministration, **C**ommerce and **T**ransport (http://www.unece.org/trade/untdid/welcome.htm)
W3C	**W**orld **W**ide **W**eb **C**onsortium (http://www.w3c.org)
WYSIWYG	**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et
XML4DR	**XML** for **D**igital **R**eporting

Reference

1:]init[AG, Berlin, Standards and Architectures for e-Government Applications v 1.1, 2003
2: Bruce Perens, Sincere Choice, 2003, http://sincerechoice.org/
3: The World Wide Web Consortium, Extensible Markup Language (XML), 2003, http://www.w3.org/XML/
4: Anders Tornqvist (Comfact AB), Specification of RDRMES XML Version 0.1 (Draft: 2002-01-07), 2002
5: W3C, HTML 4.01 Specification, 1999, http://www.w3.org/TR/html4/
6: DOM Working Group, Document Object Model (DOM), 2003, http://www.w3.org/DOM/
7: David Flanagan, Javascript, The Definitive Guide, 4. edition, 2002
8: Danny Goodman, Dynamic HTML, The Definitive Reference, 2nd Edition, 2002
9: Bert Bos (W3C) & Håkon Wium Lie (W3C) & Chris Lilley (W3C) & Ian Jacobs (W3C), Cascading Style Sheets, level 2 CSS2 Specification, 1998, http://www.w3.org/TR/REC-CSS2/
10: The Math Working Group (W3C), MathML, 2003, http://www.w3.org/Math/
11: Uwe Kunzler, Eurostat, XML standardisation for date collection and exchange, 2002
12: , edifact-xml project homepage, , http://www.edifact-xml.org/
13: Micah Dubinko (Cardiff Software) & Leigh L. Klotz, Jr. (Xerox Corporation) & Roland Merrick (IBM) & T. V. Raman (IBM), XForms 1.0, 2002, http://www.w3.org/TR/2002/CR-xforms-20021112/
14: Doug Tidwell, XSLT, 2001
15: David FlanaganJavascript, The Definitive Guide, 4. edition ,Chapter 17.1.5.1 , 2002
16: US Office of Personnell Management, , 2030, http://www.opm.gov/forms/html/formflow.asp
17: Michael Koehne, XML::Edifact - an approach towards XML/EDI as a prototype in perl, v0.40, , http://www.xml-edifact.org/TR/XML-Edifact-1.html
18: Joint Rapporteurs Message Design Group JM8, United Nations Directories for Electronic Data Interchange forAdministration, Commerce and Transport, 2000, http://www.unece.org/trade/untdid/d00a/trmd/rdrmes_c.htm
19: Anders Tornqvist & Mats Johnsson, User Guide for IQML XML Version 2.0, 2002
20: Opera Software, Web Specifications Supported in Opera 7, 2003, http://www.opera.com/docs/specs/
21: Håkon W Lie, Formatting Objects considered harmful, 1999, http://people.opera.com/howcome/1999/foch.html

22: , PNG (Portable Network Graphics) SpecificationVersion 1.0, , PNG (Portable Network Graphics) SpecificationVersion 1.0
23: Aaron Adams, Proper PNG Support in Internet Explorer for Windows, 2003, http://www.petitiononline.com/msiepng/petition.html
24: M.H. Snaprud & N. Ulltveit-Moe & O.C. Granmo & M.E. Rafoshei-Klev & A. Wiklund & A. Sawicka, Accessibility to the Internet, 2003
25: Danny GoodmanDynamic HTML, The Definitive Reference, 2nd Edition ,Chapter 2 , 2002
26: Copyright Opera Software ASA, DOM 2 Core, 2003, http://www.opera.com/docs/specs/opera6/js/dom-core.dml
27: Microsoft Corp., Internet Exploerer 6 and Standards, 2003, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dndude/html/dude03262001.asp
28: Greg Roelofs (libpng.org), Browsers with PNG Support , , http://www.libpng.org/pub/png/pngapbr.html
29: James Maguire, Microsoft Unveils Licensing Discount To Couter Linux, 2002, http://www.newsfactor.com/perl/story/20105.html
30: Ralph Naider, Speech: Resisting "Empire", Affirming Our Vision, 2003

Appendix A - A DOM example, loading an external XML file

The underlying code, was Microsoft DOM, the entire script was rewritten, to make it W3C DOM compliant.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta content=
"HTML Tidy for Linux/x86 (vers 1st March 2003), see www.w3.org"
name="generator">
<script type="text/javascript;version=1.5"><!--
var xmlDoc='';

function start(){
  /*Testing if the DOM function is implemented. The hasFeature() asks if
    the hole module is implemented. But the module can be partial implemented.*/
  if (document.implementation && document.implementation.createDocument){
    //The DOM way of loding of a xml-document
    xmlDoc=  document.implementation.createDocument("","doc",null);
    xmlDoc.addEventListener('load', afterinit , false);
  }
  else if (window.ActiveXObject){
    //Microsoft private way of loading file
    alert('Your browser do not support fileloading in W3C DOM,'
        +'trying Microsoft ActiceX');
    xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
    xmlDoc.async=false; //Enforce download of XML file first. IE only.
    afterinit();
  }
  if (typeof xmlDoc!="undefined"){
    xmlDoc.load("note.xml");
  }
}
function afterinit(){
  node = xmlDoc.getElementsByTagName('note').item(0);
  /*Get the value from the xml document and write them
    in to the HTML document.*/
  document.getElementById('to').appendChild
    (document.createTextNode
     (node.getElementsByTagName('to').item(0).firstChild.nodeValue));
  document.getElementById('from').appendChild
    (document.createTextNode
     (node.getElementsByTagName('from').item(0).firstChild.nodeValue));
  document.getElementById('header').appendChild
    (document.createTextNode
     (node.getElementsByTagName('heading').item(0).firstChild.nodeValue));
  document.getElementById('body').appendChild
    (document.createTextNode
     (node.getElementsByTagName('body').item(0).firstChild.nodeValue));
}
--></script>
<title>HTML using XML data</title>
</head>
```

```
50: <body bgcolor="yellow" onload="start()">
51: <h1>W3Schools Internal Note</h1>
52: <b>To:</b> <span id="to"></span> <br>
53: <b>From:</b> <span id="from"></span>
54: <hr>
55: <b><span id="header"></span></b>
56: <hr>
57: <span id="body"></span>
58: </body>
59: </html>
```

Appendix B - MSIE code to appendix A

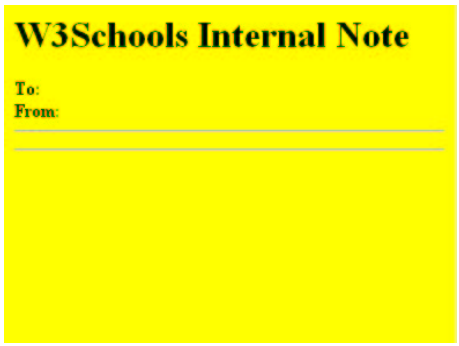
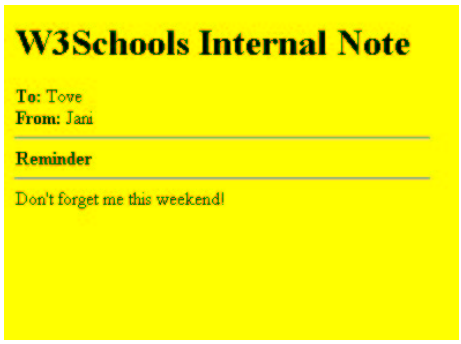
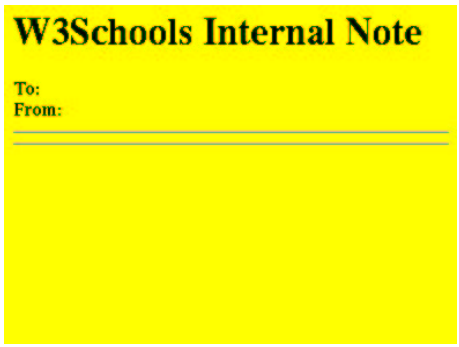
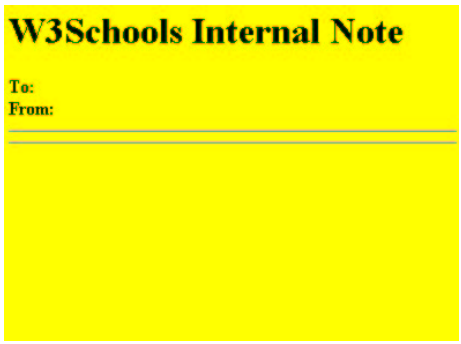
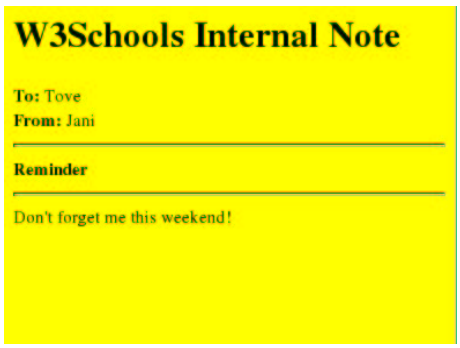
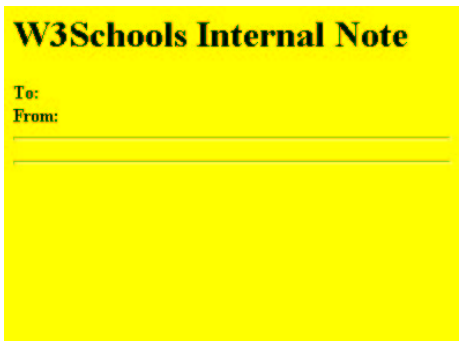
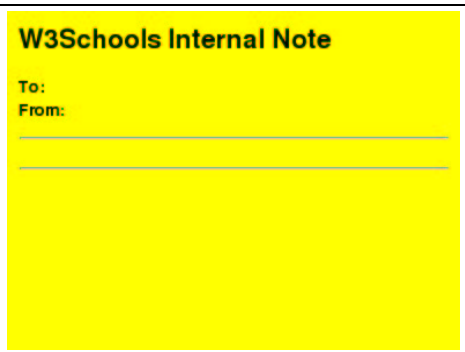
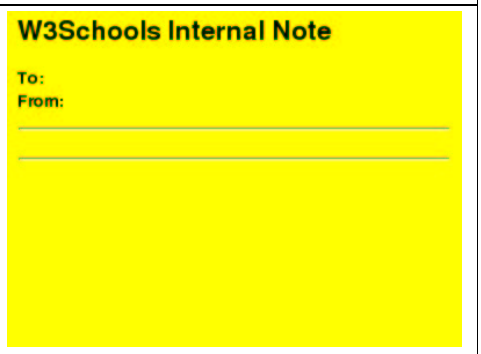
The underlying code is not W3C DOM. This is MSIE only code, the script and the HTML is not according to standard.

```
 1: <html>
 2: <head>

 4: <script type="text/javascript" for="window" event="onload">
 5: var xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
 6: xmlDoc.async="false";
 7: xmlDoc.load("note.xml");
 8: nodes = xmlDoc.documentElement.childNodes;
 9: to.innerText = nodes.item(0).text;
10: from.innerText = nodes.item(1).text;
11: header.innerText = nodes.item(2).text;
12: body.innerText = nodes.item(3).text;
13: </script>

15: <title>HTML using XML data</title>
16: </head>

18: <body bgcolor="yellow">

20: <h1>W3Schools Internal Note</h1>

22: <b>To: </b>
23: <span id="to"> </span>
24: <br>

26: <b>From: </b>
27: <span id="from"></span>
28: <hr>

30: <b><span id="header"></span></b>

32: <hr>
33: <span id="body"></span>

35: </body>
36: </html>
```

Appendix C - xml-save func. for App. A

```
 1: function putXmlFile(){
 2: /*****************************************************
 3:   * This functions uploads the xmlDoc on to a web      *
 4:   * Server. The functions works with Appendix A        *
 5:   * The function to not ask if the browser support     *
 6:   * the needed standard. It just try to run the code   *
 7:   * and tries the next, if the first fails.            *
 8:   *                                                    *
 9:   * The function can be pasted into Appendix A,        *
10:   * and run.                                           *
11:   *****************************************************
12:  */
13:   var xmlHttp=false;
14:   if (document.implementation
15:       && document.implementation.hasFeature
16:       && document.implementation.hasFeature('HTML','2.0')){
17:     var tmpCursor=document.body.style.cursor;
18:     document.body.style.cursor='wait';
19:   }
20:   try {
21:     xmlHttp = new ActiveXObject ('Msxml2.XMLHTTP');
22:   }catch (e){
23:     try{
24:       var xmlHttp = new ActiveXObject ('Microsoft.XMLHTTP');
25:     }catch (E)
26:     {xmlHttp=false;
27:     }
28:     if (!xmlHttp){
29:       var xmlHttp = new XMLHttpRequest();
30:     }

32:     var uri='http://localhost/upload/';
33:     var uloadFileName='loadandsave.xml';
34:     try {
35:       xmlHttp.open("PUT",uri+uloadFileName);
36:     }catch (e){alert(e);
37:       xmlHttp.onreadystatechange=function() {
38:         if (xmlHttp.readyState==4){
39:           alert (xmlHttp.responseText);
40:         }
41:       }

43:       xmlHttp.setRequestHeader("Man", 'PUT ' + uri + uloadFileName);
44:       xmlHttp.setRequestHeader("Content-Type", "text/xml");

46:       xmlHttp.send (xmlDoc);
47:     }

49:     if (document.implementation
50:         && document.implementation.hasFeature
51:         && document.implementation.hasFeature('HTML','2.0')){
52:       document.body.style.cursor=tmpCursor;
53:     }
54:   }
```

Appendix D - Output from App. A and B

Browser	*W3C DOM code*	*Microsoft DOM code*
MSIE		
Opera		
Mozilla		
KHTML		

Appendix E - hasFeature() results

Module	*MSIE 6.5*			*Mozilla 1.3 and 1.4a1*			*Opera 7.1*			*KHTML 3.1*			*Notes*
	L1	L2	L3	L1	L2	L3	L1	L2	L3	L1	L2	L3	
Core	-	N	N	-	**Y**	N	-	N	N	-	N	N	
XML	**Y**	N	N	**Y**	**Y**	N	N	N	N	Y	N	N	
HTML	N	N	n/a	**Y**	**Y**	n/a	**Y**	N	n/a	Y	N	n/a	
Views	n/a	N	n/a	n/a	**Y**	n/a	n/a	N	n/a	n/a	N	n/a	
Style Sheets	n/a	N	n/a	n/a	**Y**	n/a	n/a	N	n/a	n/a	N	n/a	
CSS	n/a	N	n/a	n/a	**Y**	n/a	n/a	N	n/a	n/a	N	n/a	
CSS2	n/a	N	n/a	n/a	**Y**	n/a	n/a	N	n/a	n/a	N	n/a	
Events	n/a	N	N	n/a	**Y**	N	n/a	**Y**	N	n/a	N	N	
User interface Events	n/a	N	N	n/a	N	N	n/a	**Y**	N	n/a	N	N	
Mouse Events	n/a	N	N	n/a	**Y**	N	n/a	**Y**	N	n/a	N	N	
Text Events	n/a	n/a	N	n/a	n/a	N	n/a	n/a	N	n/a	n/a	N	
Mutation Events	n/a	N	N	n/a	N	N	n/a	**Y**	N	n/a	N	N	
HTML Events	n/a	N	N	n/a	**Y**	N	n/a	**Y**	N	n/a	N	N	
Range	n/a	N	n/a	n/a	**Y**	n/a	n/a	N	n/a	n/a	N	n/a	
Traversal	n/a	N	n/a	n/a	N	n/a	n/a	N	n/a	n/a	N	n/a	
Load and Save	n/a	n/a	N	n/a	n/a	N*	n/a	n/a		n/a	n/a	N	* load() implemented
Abstract Schemas Editing	n/a	n/a	N	n/a	n/a	N	n/a	n/a		n/a	n/a	N	
XPath	n/a	n/a	N	n/a	n/a	**Y**	n/a	n/a		n/a	n/a	N	

Appendix F - hello world source code

```
 1: /*file: ----------------------------------------------------------------xml4dr.js*/
 2: /*   (c) Michael Eric Menk                                              */
 3: /*   Public domain (all code here is trivial)                           */
 4: /*   (Exept functions with other notes)                                 */
 5: /*-------------------------------------------------------------------Globals_vars*/
 6: var xmlDocument;
 7: var language='no';
 8: var uloadFileName='xml4dr_upload.xml';
 9: //gets Navigators default language.
10: if (language=navigator.language) {
11:   language=navigator.language[0];
12: }
13: var xmlDoc_textElem;
14: var tableElement='';
15: var xmlDoc_globTmpElem;
16: var htmlDoc_globTmpElem;
17: var xmlDoc='';

19: /*-----------------------------------------------------------------appendLogg(text)*/
20: // This is a debug logger. It writes out a logg at the end of the document.
21: function appendLogg(text){
22:   //comment remove return, to get more logging
23:   return;

25:   if (document.implementation.hasFeature('HTML','1.0')){
26:     currentID= document.getElementById('log'); //DOM HTML
27:   }
28:   else{ // The browser do not support DOM Level 1 HTML
29:   currentID= document.getElementsByTagName('ul');
30:   for ( i=0 ; currentID[i] ;  i++ ){
31:     if ( currentID[i].getAttribute('id') == 'log' )
32:     {
33:       currentID=currentID[i];
34:       break;
35:     }
36:   }
37:   }
38:   currentID=currentID.appendChild(document.createElement('li'));
39:   currentID=currentID.appendChild(document.createTextNode(text));
40: }
41: /*-----------------------------------------------------------appendErrorLogg(text)*/
42: //Same as appendLogg, just its errors, and the text is of type <strong>
43: function appendLoggError(text){

45:   currentID= document.getElementById('log');
46:   currentID=currentID.appendChild(document.createElement('li'));
47:   currentID=currentID.appendChild(document.createElement('strong'));
48:   currentID.appendChild( document.createTextNode(text));
49: }
50: /*function-------------------------------------------createXMLFromString (string)*/
51: /* This function crates a XML document from a string. This is for the
52:    fall back. The function is also run when a XML file is loaded, this
53:    was to make 100% sure that i did not have any ActiveX functions.
54:    Just to be on the safe side.
55:    */
```

```
 56: function createXMLFromString (string) {
 57:   /*
 58:      ************************************************************
 59:      *  Function source:  http://www.faqts.com                  *
 60:      *                                                          *
 61:      ************************************************************
 62:    */
 63:   appendLogg('Enter function: createXMLFromString');
 64:   var xmlParser;
 65:   try {
 66:     xmlParser = new DOMParser();
 67:     xmlDocument = xmlParser.parseFromString(string, 'text/xml');
 68:     return xmlDocument;
 69:   }catch (e) {
 70:     alert("Can't create XML document.");
 71:     return null;
 72:   }
 73: }
 74: /*function------------------------------------------- serializeXML (xmlDocument)*/
 75: /* This function converts the XML document to a text string. This is used to
 76:    print out out the xml file on the web page, and used when the xml document is
 77:    http-put on to a server*/
 78: function serializeXML (xmlDocument) {
 79:   /*
 80:      ************************************************************
 81:      *  Function source:  http://www.faqts.com                  *
 82:      *                                                          *
 83:      ************************************************************
 84:    */
 85:   appendLogg('Enter function: serializeXML ');
 86:   var xmlSerializer;
 87:   try {
 88:     xmlSerializer = new XMLSerializer(); //Opera 7.1 do not support this (DOM L2)
 89:     return xmlSerializer.serializeToString(xmlDocument);
 90:   }catch (e) {
 91:     alert("Can't serialize XML document.");
 92:     return '';
 93:   }
 94: }
 95: /*-------------------------------------------------------------actionMouseTableOut()*/
 96: /* This function is run when the mouse leaves the html table that where the form
 97:    is printed. The function updates the XMLDoc and then runs the
 98:    actionMouseTableOut() to update the xml file on the web page*/
 99: function actionMouseTableOut(){
100:   var cHtml='';
101:   var cXml='';
102:   var xmlTagsData='';
103:   //  appendLogg('Enter function: actionMouseTableOut');

105:   if (!xmlDocument) appendLoggError(' xmlDOC is empty in actionMouseTableOut');

107:   try{
108:     //Gets all data elements from the XML file
109:     xmlTagsData=xmlDocument.getElementsByTagName('Data');
110:   }catch (e){
111:     appendLoggError(e);
112:   }
113:   //traverse and update all the data, and their html counterparts.
```

```
114:   for (var i=0 ; xmlTagsData[i]  ;  i++ ){

116:     cHtml=document.getElementById('xmlData_' +
117:         xmlTagsData[i].getAttribute('dataId'));

119:     try{
120:       if (!xmlTagsData[i].hasChildNodes()){
121:         xmlTagsData[i].appendChild(document.createTextNode(cHtml.value));
122:       }else{
123:         xmlTagsData[i].firstChild.nodeValue=cHtml.value;
124:       }

126:       appendLogg('New xml node value:' + xmlTagsData[i].firstChild.nodeValue);
127:     }catch (e){
128:       appendLoggError( e );
129:     }
130:   }
131:  update_Id_xml_after();
132: }

134: /* function-------------------------------------------------------createHTMLTable()*/
135: /* This functions generates the html table. The functions traverse the XML
136:    document.
137:    */
138: function createHTMLTable(){
139:   appendLogg('Enter function: createHTMLTable');
140:   if (xmlDocument.documentElement.nodeName != 'Rdrmes') {
141:     alert("Not correct XML-file");
142:     appendLoggError('The XML-file have a wrong nodeName');
143:   }else{
144:     //loads text objects

146:     var id='dominsert';
147:     var borderw=1;
148:     var tmpElement;
149:     var tableElement;
150:     var currentElement=document.getElementById(id);
151:     currentElement=currentElement.appendChild(document.createElement('form'));
152:     currentElement.setAttribute('id','Form_'
153:         + xmlDocument.documentElement.getAttribute('formId'));
154:     currentElement=currentElement.appendChild(document.createElement('table'));
155:     currentElement.setAttribute('border',borderw);
156:     currentElement.setAttribute('id','Table_'
157:         + xmlDocument.documentElement.getAttribute('formId'));
158:     currentElement.setAttribute('onmouseout','actionMouseTableOut();');
159:     tableElement=currentElement;

161:     var xmlDocElm_1= xmlDocument.documentElement;

163:     var nodeindex=0;
164:     while ( xmlDocElm_1.childNodes[nodeindex] ){
165:       if ( xmlDocElm_1.childNodes[nodeindex].nodeType==1 ){
166:         swhich ( xmlDocElm_1.childNodes[nodeindex].nodeName ){
167:           case 'TextRef':
168:             {
169:               if (
getTextType(xmlDocElm_1.childNodes[nodeindex].firstChild.nodeValue)=='Heading' ){
170:                 //Add <tr><td><b><big>getText()</big></b></td></tr>
```

```
171:                 currentElement=currentElement.appendChild
172:                   (document.createElement('tr'));
173:                 currentElement=currentElement.appendChild
174:                   (document.createElement('td'));
175:                 currentElement.setAttribute('colspan','2' );
176:                 currentElement=currentElement.appendChild
177:                   (document.createElement('center'));
178:                 currentElement=currentElement.appendChild
179:                   (document.createElement('strong'));
180:                 currentElement=currentElement.appendChild
181:                   (document.createElement('big'));
182:                 currentElement.appendChild(document.createTextNode
183:                     (getTextString
184:                      (xmlDocElm_1.childNodes[nodeindex].firstChild.nodeValue)));
185:                 currentElement=tableElement;
186:               }else if (
getTextType(xmlDocElm_1.childNodes[nodeindex].firstChild.nodeValue)=='Instruction' ){
187:                 //
188:               }else { alert('ERROR: never claime: textID error'); }
189:               break;
190:             }
191:           case 'Instance':
192:             {
193:               htmlDoc_globTmpElem=tableElement;
194:               renderXmlTagInstance(xmlDocElm_1.childNodes[nodeindex],tableElement);
195:               break;
196:             }
197:           case 'Text':
198:             {
199:               //Text is treated elseware. skip only
200:               break;
201:             }
202:         }
203:       }
204:       nodeindex++;
205:     }
206:   }

208:   currentElement=tableElement.appendChild(document.createElement('tr'));
209:   currentElement=currentElement.appendChild(document.createElement('td'));

211:   currentElement.appendChild(document.createTextNode
212:       ('Insert upload file name'));
213:   currentElement
214:     =currentElement.parentNode.appendChild(document.createElement('td'));
215:   currentElement=currentElement.appendChild(document.createElement('input'));
216:   currentElement.setAttribute('type','text');
217:   currentElement.setAttribute('id','text_ul_fname' );
218:   currentElement.setAttribute('name','text_ul_fname');
219:   currentElement.setAttribute('value', uloadFileName );
220: }

222: //if (document.implementation.createDocument){
appendLoggError('document.implementation.createDocument not exist');}
223: var xmlDoc = document.implementation.createDocument('', 'test', null);
224: xmlDoc.addEventListener('load', docFine, false);

226: function docFine(){
```

```
227:   var s = new XMLSerializer();
228:   var tmpstr = s.serializeToString(xmlDoc);

230:   tmpXmlFile=createXMLFromString ( tmpstr );
231:   //Test to se if the loaded document fist node, is a valid name
232:   if ( tmpXmlFile.documentElement.nodeName == 'Rdrmes' ){
233:     xmlDocument=tmpXmlFile;
234:   }

236:   afterLoadXmlFile();
237: }
238: function loadXmlFile()
239: {
240:   appendLogg( 'Entering function loadXmlFile' );
241:  //fallback
242:   xmlDocument = createXMLFromString(
243:       '<?xml version="1.0" encoding="UTF-8"?>'
244:       +'<TextRef>text01</TextRef>'
245:       + 'text01'
246:       +  '<Instance instanceId="i1">'
247:       +   '<Set setId="s1">'
248:       +    '<Domain domainId="d01"></Domain>'
249:       +    '<Tuple tulple="t01">'
250:       +     '<TextRef>text02</TextRef>'
251:       +     '<Data dataId="data01d01">'
252:       +       '--=|Fallback, this file was not loaded from file..|=--'
253:       +     '</Data></Tuple></Set>'
254:       +  '</Instance>'
255:       +  '<Text textId="text01" textType="Heading">'
256:       +   '<textString language="en">Hello World!</textString>'
257:       +   '<textString language="no">Hallo Verden!</textString>'
258:       +  '</Text>'
259:       +  '<Text textId="text02" textType="Instruction">'
260:       +   '<textString language="en">Please enter your name:</textString>'
261:       +   '<textString language="no">Skriv in navnet ditt:</textString>'
262:       + '</Text>'
263:       +'</Rdrmes>');

265:   xmlDoc.load("xml4dr.xml");
266: }

268: function afterLoadXmlFile()
269: {
270:   if (xmlDocument) {
271:     //gets ref. to text elements.
272:     xmlDoc_textElem=xmlDocument.getElementsByTagName('Text');
273:     appendToID('---XML-file_before---','xml_before');
274:     appendToID('XML source: ' + serializeXML(xmlDocument) , 'xml_before');

276:     clearHtmlId('dominsert');
277:     createHTMLTable();

279:     //update xml-document (needs to bee updated when actions are preformed)
280:     appendToID('---XML-file_after---','xml_after');
281:     appendToID('XML source: ' + serializeXML(xmlDocument) , 'xml_after');
282:   }
283: }
284: // The main function, is called on on load by the html doc.
```

```
285: /* function-------------------------------------------------------------init()*/
286: function init(){
287:   appendLogg('Start');
288:   try{
289:   var s=new XMLSerializer();
290:   }catch (e){
291:     appendLoggError(e);
292:   }
293:   loadXmlFile();
294: }
295: function putXmlFile(){

297:   var xmlHttp=false;
298:   if (document.implementation.hasFeature('HTML','2.0')){
299:     var tmpCursor=document.body.style.cursor;
300:     document.body.style.cursor='wait';
301:   }
302:   try {
303:     xmlHttp = new ActiveXObject ('Msxml2.XMLHTTP');
304:   }catch (e){
305:     try{
306:       var xmlHttp = new ActiveXObject ('Microsoft.XMLHTTP');
307:     }catch (E)
308:     {xmlHttp=false;
309:     }
310:     if (!xmlHttp){
311:       var xmlHttp = new XMLHttpRequest();
312:     }


315:     var uri='http://localhost/upload/';
316:     var uloadFileName='xml4dr_put.xml';
317:
318:
319:     uloadFileName=document.getElementById('text_ul_fname').value;
320:     try {
321:     xmlHttp.open("PUT",uri+uloadFileName);
322:     }catch(e){alert(e);}
323:     xmlHttp.onreadystatechange=function() {
324:       if (xmlHttp.readyState==4){
325:         alert (xmlHttp.responseText);
326:       }
327:     }

329:     xmlHttp.setRequestHeader("Man", 'PUT ' + uri + uloadFileName);
330:     xmlHttp.setRequestHeader("Content-Type", "text/xml");

332:     xmlHttp.send (xmlDocument);
333:   }

335:   if (document.implementation.hasFeature('HTML','2.0')){
336: //    document.body.style.cursor=tmpCursor;
337:     document.body.style.cursor='default';
338:   }
339: }
```

```
343: /*-----------------------------------------------------renderXmlTagInstance()*/
344: function renderXmlTagInstance (xmlElm,htmlElm) {
345:   appendLogg('Enter function: renderXmlTagInstance');
346:   if (xmlElm)  appendLogg('>>> xmlElm.nodeName: ' + xmlElm.nodeName);
347:   if (htmlElm) appendLogg('>>> htmlElm.nodeName: ' + htmlElm.nodeName);

349:   //xmlElm = base element of this funtion. (det Set tag)
350:   //htmlElm = the HTMl element where to do the update (a <tr>)

352:   var currentHtml=htmlElm;
353:   var nodeindex=0;
354:   while ( xmlElm.childNodes[nodeindex]){
355:     if (xmlElm.childNodes[nodeindex].nodeType==1 /*ELEMENT_NODE-*/ ){
356:       appendLogg ('Childnode: ' + xmlElm.childNodes[nodeindex].nodeName);
357:       swhich (xmlElm.childNodes[nodeindex].nodeName){
358:         case 'CatchEvent':
359:           {
360:             appendLogg('>>> case: Instance/CatchEvent');
361:             //not implemented
362:             break;
363:           }
364:         case 'TextRef':
365:           {
366:             appendLogg('>>> case: Instance/TextRef');
367:             //--insert text
368:             break;
369:           }
370:         case 'Set':
371:           {
372:             appendLogg('>>>> case: Instance/Set');
373:             renderXmlTagSet(xmlElm.childNodes[nodeindex],currentHtml);
374:             //--insert case
375:             break;
376:           }
377:         case 'Note':
378:           {
379:             appendLogg('>>>> case: Instance/Note');
380:             //--Note not implemented
381:             break;
382:           }

384:       }
385:     }
386:     nodeindex++;
387:   }

389: }

391: /*function ------------------------------------renderXmlTagSet(xmlElm,htmlElm)--*/
392: function renderXmlTagSet(xmlElm,htmlElm) {

394:   appendLogg('Enter function: renderXmlTagSet');
395:   if (xmlElm)  appendLogg('>>> xmlElm.nodeName: ' + xmlElm.nodeName);
396:   if (htmlElm) appendLogg('>>> htmlElm.nodeName: ' + htmlElm.nodeName);
397:   //xmlElm = base element of this funtion. (det Set tag)
398:   //htmlElm = the HTMl element where to do the update (a <tr>)

400:   var nodeindex=0;
```

```
401:   while ( xmlElm.childNodes[nodeindex]){
402:     if (xmlElm.childNodes[nodeindex].nodeType==1 || true){
403:       swhich (xmlElm.childNodes[nodeindex].nodeName){
404:         case 'CatchEvent':
405:           {
406:             appendLogg('>>>> case: Instance/Set/CatchEvent :: nodeindex='
407:                 +nodeindex);
408:             //--Not Implemented
409:             break;
410:           }
411:         case 'TextRef':
412:           {
413:             appendLogg('>>>> case: Instance/Set/TextRef :: nodeindex='+nodeindex);
414:             //--insert text
415:             break;
416:           }
417:         case 'Domain':
418:           {
419:             appendLogg('>>>> case: Instance/Set/Domain :: nodeindex=' + nodeindex);
420:             //--Not Implementd
421:             break;
422:           }
423:         case 'Tuple':
424:           {
425:             appendLogg('>>>> case: Instance/Set/Tuple :: nodeindex=' + nodeindex);
426:             renderXmlTagTuple (xmlElm.childNodes[nodeindex],htmlElm);
427:               //--insert tuple code
428:             break;
429:           }

431:       }
432:     }
433:     nodeindex++;
434:   }
435: }

437: /*function ----------------------------------renderXmlTagTuple (xmlElm,htmlElm)--*/
438: function renderXmlTagTuple (xmlElm,htmlElm) {

440:   appendLogg('Enter function: renderXmlTagTuple');
441:   if (xmlElm)  appendLogg('>>> xmlElm.nodeName: ' + xmlElm.nodeName);
442:   if (htmlElm) appendLogg('>>> htmlElm.nodeName: ' + htmlElm.nodeName);
443:   //xmlElm = base element of this funtion. (det Set tag)
444:   //htmlElm = the HTMl element where to do the update (a <tr>)

446:   var tmpHtml='';
447:   var cHtml=htmlElm;
448:   var cXmlElm='';
449:   var tmpTextRef='';
450:   var nodeindex=0;

452:   if (cHtml.nodeName=='TABLE'){
453:     cHtml=cHtml.appendChild(document.createElement('tr'));
454:     appendLogg('>>>HTML>  '+cHtml.nodeName+' created' );
455:   }

457:   baseHtmlTableTr=cHtml;
```

```
459:   while ( xmlElm.childNodes[nodeindex]){
460:     if (xmlElm.childNodes[nodeindex].nodeType==1 || true){
461:       swhich (xmlElm.childNodes[nodeindex].nodeName){
462:         case 'TextRef':
463:           {
464:             appendLogg('>>>> case: Instance/Set/Tuple/TextRef :: nodeindex='
465:                 + nodeindex);
466:             cXmlElm=xmlElm.childNodes[nodeindex];
467:             tmpTextRef= cXmlElm.firstChild.nodeValue;

469:             if (cHtml.nodeName=='TR'){
470:               cHtml=cHtml.appendChild(document.createElement('td'));
471:               appendLogg('>>>HTML>  '+cHtml.nodeName+' created' );
472:             }
473:             if(getTextType(tmpTextRef)=='heading'){
474:               cHtml.appendChild(document.createElement('strong'));
475:             }
476:             cHtml.appendChild(htmlPrintTextRef(tmpTextRef));
477:               appendLogg('>>>HTML>  Text appended' );

479:             //-- insert textref code
480:             break;
481:           }
482:         case 'Data':
483:           {
484:             appendLogg('>>>> case: Instance/Set/Tuple/Data :: nodeindex='
485:                 + nodeindex);

487:             cHtml=baseHtmlTableTr;

489:             cXmlElm=xmlElm.childNodes[nodeindex];

491:             if (cHtml.nodeName=='TR'){
492:               cHtml=cHtml.appendChild(document.createElement('td'));
493:               appendLogg('>>>HTML>  '+cHtml.nodeName+' created' );
494:             }

496:             tmpHtml=cHtml.appendChild(document.createElement('input'));
497:             tmpHtml.setAttribute('type','text');
498:             tmpHtml.setAttribute('id','xmlData_' + cXmlElm.getAttribute('dataId'));
499:             tmpHtml.setAttribute('name','xmlData_'
500:                 + cXmlElm.getAttribute('dataId'));
501:             tmpHtml.setAttribute('size','30');
502:             //The next line chage the xml-doc when writing, Not needed.
503:             tmpHtml.setAttribute('onchange','actionMouseTableOut()');
504:             if (cXmlElm.firstChild)
505:               tmpHtml.setAttribute('value',cXmlElm.firstChild.nodeValue);
506:             appendLogg('From text field added: id=' + tmpHtml.getAttribute('id'));
507:             //type="text" name="test" size="20" value="value"
508:             //-- insert date code
509:           }
510:       }
511:     }
512:     nodeindex++;
513:   }

515: }
```

```
519: /*function --TEMPLATE--------------------------renderXmlTagSet(xmlElm,htmlElm)--*/
520: /*  -- template of tag traverce.
521:     function renderXmlTagSet(xmlElm,htmlElm) {
522:     appendLogg('Enter function: ');
523:     if (xmlElm)  appendLogg('>>> xmlElm.nodeName: ' + xmlElm.nodeName);
524:     if (htmlElm) appendLogg('>>> htmlElm.nodeName: ' + htmlElm.nodeName);
525: //xmlElm = base element of this funtion. (det Set tag)
526: //htmlElm = the HTMl element where to do the update (a <tr>)
527: alert ('Set!!');


530: var nodeindex=0;
531: while ( xmlElm.childNodes[nodeindex]){
532: if (xmlElm.childNodes[nodeindex].nodeType==1){
533: swhich (xmlElm.childNodes[nodeindex].nodeName){
534: }
535: }
536: nodeindex++;
537: }

539: }

541: -- template fine */
542: /*function---------------------------------------htmlPrintTextRef(TextRefString)*/
543: // A small function to make the command smaller. Returns a new TextNode
544: function htmlPrintTextRef(TextRefString)
545: {
546:  return document.createTextNode(getTextString(TextRefString));
547: }

549: /*function-----------------------------------------------------appendToBody(text)*/
550: // Adds lines to the end of the html-doc.
551: function appendToID (text,id) {
552:   if (document.body && document.createTextNode) {
553:     var currentID = document.getElementById(id);
554:     currentID.appendChild(document.createElement('br'));
555:     currentID.appendChild(document.createTextNode(text));
556:   }
557: }
558: function clearHtmlId(id){
559:   document.getElementById(id).innerHTML="";
560: }
561: /*function ----------------------------------------update_Id-xml_after()--*/
562: // write the xml file in the html document
563: function update_Id_xml_after(){
564:   clearHtmlId("xml_after");
565:   appendToID("---XML-file_after---\n","xml_after");
566:   appendToID('XML source: ' + serializeXML(xmlDocument) , "xml_after");
567: }
568: /*function ----------------------------------------------------getTextType()--*/
569: // Returns the text type from the xml4dr file. And returns the string
570: function  getTextType (textId){
571:   appendLogg('Enter function: getTextType ');
572:   for ( var i=0 ; xmlDoc_textElem[i] ; i++ ){
573:     if (xmlDoc_textElem[i].getAttribute('textId') == textId )
574:     {
```

```
575:       var type=xmlDoc_textElem[i].getAttribute('textType');
576:     }
577:   }
578:   return type;
579: }

581: /*function ------------------------------------------------------------getTextId()--*/
582: // returns the text, from the xml4dr file
583: function  getTextString (textId){
584:   appendLogg('Enter function: getTextString ');
585:   var match='';
586:   var tmpElement='';
587:   // finds the textelement in question
588:   for ( var i=0 ; xmlDoc_textElem[i] ; i++ ){
589:     if ( xmlDoc_textElem[i].getAttribute('textId') == textId ) {match=i;}
590:   }

592:   // Set tmp Element to to the first element (in other words, the firs language.
593:   tmpElement=xmlDoc_textElem[match].getElementsByTagName('textString');
594:   if ( !tmpElement[0] ){alert('ERROR: Nerver claim in function getTextString:
tmpElement empty ' );}

596:   if ( tmpElement[0].firstChild.nodeType==3 /*text (not supported by MSIE)*/){
597:     var textString=tmpElement[0].firstChild.nodeValue;
598:   }else{
599:     alert('ERROR: Never claim in function getTextString, wrong node type');
600:   }

602:   // Change the tmpElement if your language is listed.
603:   for ( var i=0 ; tmpElement[i] ; i++ ){
604:     if ( tmpElement[i].getAttribute('language') == language ){
605:       if ( tmpElement[i].firstChild.nodeType==3 /*text (not supported by MSIE)*/){
606:         textString=tmpElement[i].firstChild.nodeValue;
607:       }else{
608:         alert('ERROR: Never claim in function getTextString, wrong node type');
609:       }
610:     }
611:   }
612:   return textString;
613: }
```

Appendix G - Software used in this thesis.

- Editors: Vim 6.1 (Source code), and OpenOffice.org (This document)
- Browsers [for testing of the code]:
  - Mozilla 1.3 and 1.4a in GNU/Linux 2.4 and MS-Windows 2000 (NT 5)
  - Opera 7.1 in GNU/Linux 2.4 and MS-Windows 2000 (NT 5)
  - MS Internet Explorer in MS-Windows 2000 (NT 5)
  - Konqueror (KHTML engine) in GNU/Linux 2.4
- Utilities
  - Tidy [for cleaning HTML and XML docs]
  - Festival [for reading this document out loud]
  - java2html [Syntax highlighting of thecode]