Vrije Universiteit Brussel
Faculteit Wetenschappen
Departement Informatica

Het gebruik van XML bij het modelleren van Conceptuele Schema in Website Design

Tom Decruyenaere
Academiejaar
1998-1999

Verhandeling tot het behalen van de graad van
Licentiaat in de Toegepaste Informatica

Promotor : Prof. Olga De Troyer

Vrije Universiteit Brussel
Faculteit Wetenschappen
Departement Informatica

The Use of XML in modeling Conceptual Schemas in Website Design

Tom Decruyenaere
Academy year
1998-1999

Essay brought forward to achieve the degree
of Licentiate in the Applied Computer Science

Promoter : Prof. Olga De Troyer

Samenvatting

Internet neemt in steeds toenemende mate een belangrijke rol in binnen onze informatie maatschappij. Daar waar enkele jaren geleden het Internet niet meer was dan een experiment binnen een wetenschappelijke omgeving is het nu een der belangrijkste informatiebronnen in alle gelederen van onze maatschappij. De informatie op Internet wordt voorgesteld in de vorm van websites verspreid en beheerd over ontelbare servers over de wereld. De manier waarop met deze informatie wordt omgegaan evolueert dan ook gestadig naar meer professioneel gebruik waarbij het belangrijk is op een snelle en efficiënte manier toegang te krijgen tot deze data. In deze thesis hebben we geprobeerd een manier te vinden om de inhoud en de structuur van een website voor te stellen op een vast gedefinieerde en overzichtelijke wijze, in een vorm die over Internet te communiceren is. In de eerste plaats werd hiervoor gebruik gemaakt van technieken uit de Database ontwikkeling zoals conceptuele schema. Hiermee was het mogelijk in een eerste stap de verschillende bouwstenen van een website en hun onderlinge relatie op een gestructureerde manier voor te stellen in drie conceptuele subschema's. De eerste twee subschema's bevatten die elementen die respectievelijk de inhoud en de structuur voorstellen. Een derde subschema verbindt deze eerste twee. Vervolgens hebben we beroep gedaan op de mogelijkheden van XML, welke een nieuwe mark-up taal is die zijn opgang maakt in de Internet wereld. Deze taal is in tegenstelling met HTML uitbreidbaar doordat eigen tags kunnen gedefinieerd worden. Door middel van Document type definities (DTD), gekend vanuit SGML, kunnen inhoud en structuur van documenten gedefinieerd worden en op die manier onderverdeeld in verschillende klassen. In een tweede stap is een algoritme opgesteld om een conceptueel schema om te zetten in een DTD. Dit algoritme is daarna gebruikt om de gecreëerde conceptuele schema's van een website te transformeren in een DTD. Hierdoor wordt het mogelijk de inhoud en de structuur van elke instantie van een website te gaan omvormen in een XML document door gebruik te maken van de elementen gedeclareerd in de DTD. Door de definitie van een Document Object model in XML is het mogelijk om een XML document te gaan beschouwen als een object en op die manier de informatie die het bevat te gaan verwerken. In een laatste hoofdstuk is door de combinatie van dit DOM en Java script, verwerkt in HTML, aangetoond hoe een XML document van een website, die volgens de eerder bepaalde principes is opgebouwd, kan ontleed en voorgesteld worden. Op die manier is een interessant hulpmiddel gecreëerd dat bijdraagt in het efficiënt gebruik van websites op het Internet en de zoektocht naar de juiste informatie.

Abstract

Internet plays a more and more important role in our information society. A few years ago, Internet wasn't more than an experiment within a scientific environment. It is now one of the major sources of information in every rank of our society. The information on the Internet is presented in the form of websites spread over and managed by innumerable servers all over the world. The way this information is used evolves constantly to a more professional use where it is important to get access to this data in a fast and efficient way. In this thesis we have tried to find a way to represent the content and the structure of a website in a common and conveniently organized way, in a form that can be communicated over the Internet. We have used techniques from Database-development such as the conceptual schema technique. By this it was possible in a first step to represent the different building blocks of a website and their mutual relationship in a structured way in three conceptual subschemas. The first two subschemas contain those elements which represent respectively content and structure. The third subschema connects the first two. Then we have appealed to the possibilities of XML which is a new mark-up language, increasingly successful in the Internet environment. This language is -as contrasted with HTML- extendable because private tags can be defined. By means of Document Type Definitions (DTD), known from SGML, content and structure can be defined and subdivided in different classes. In a second step an algorithm meant for converting a conceptual schema into a DTD is drafted. This algorithm is then used to transform the developed conceptual schema of a website into a DTD. By this it was possible to model the content and structure of any instance of a website into an XML document using the elements declared in the DTD. By the definition of a Document Object Model (DOM) in XML it is possible to consider an XML document as an object and in that way manipulate the information it contains. In the final chapter, we have shown, by the combination of this DOM and Java Script embedded in HTML, how an XML document of a website, built up according to earlier defined principles, can be analyzed and represented.

By this means, an interesting expedient is created. It contributes to an efficient use of websites on the Internet and the search to the right information.

Acknowledgement

I would like to thank the following people for there contribution to make this thesis to what it is.

Prof. Olga De Troyer
My wife Liesbeth
My parents
My fellow student Bert Lefever
My Volvo colleagues

Table of Contents

Table of Figures

1. Definition of the Problem

1.1. The Internet

The Internet is a gigantic collection of information spread over millions of sites located on thousands of servers all over the world. This pile of data is connected together by hyperlinks making it possible to go from one website to another, jumping from one subject to the next topic. This makes the Internet the most dynamic accumulation of information but perhaps also the most disordered one. The Internet creation relies on simple techniques like HTML. This is one of the main reasons why it became as big as it is today. But because this technique doesn't have much possibilities in creating structured data presentations it also causes the fact that the Internet can be hard to explore.
The Internet has evolved from a private experiment for some scientists towards a medium that reaches millions of people all over the world. Companies have taken over these techniques and use them as a professional tool to distribute their information towards customers. In addition every big company nowadays has an intranet as an internal information provider for their employees, separating and securing it through firewalls from the Internet. All of this makes the Internet one of the most important media towards the future.
Today almost anyone who is a bit familiar with the use of a PC can go on the Internet using a browser program to look for some kind of information. To help people in their search for knowledge some applications have been developed calling themselves search engines (e.g. YAHOO, ALTAVISTA). It's hard to find a subject nowadays that won't give you a result on those search engines. The problem today is more of the kind that you will be overwhelmed with sites were your subject is mentioned and you will not see the wood for the trees. It has become very difficult to use such an enormous collection of data in an efficient way.

1.2. Problem Description

The overwhelming accessibility to data, on a global scale, does not necessarily translate to widespread utility of data. We often find that we are drowning in data, with few tools to help manage relevant data for our various activities.
The lost-in-hyper-space syndrome is a phenomena that occurs when surfing on the Internet. Generally this medium is used to find specific information and it can happen that by frequently use of references and hyperlinks in websites people loose track of where they are and what they are looking for. You can compare it with finding the way in a strange city without having a city map at your disposal. The structure and survey of a website becomes more and more important to help users to deal with their queries in an efficient way.

1.3. Objective

The objective of this thesis is to provide means to create an overview of a website that permits users to know in an efficient way what information is available inside the website and how this information is structured and reachable. We will appeal to *Conceptual Schemas* to describe the type of information present in a website and how the information is structured.
To do so, we first have to define the concept of Conceptual Schema (CS) for a website. This is done by means of a "meta" conceptual schema for websites. This meta conceptual schema defines the concepts and their relationships that will be used to define the conceptual schemas of whatever instance of a website. To model the meta conceptual schema we use the Binary Role Model (BRM) that allows for a graphical representation. Next we translate this BR-schema into an XML document type definition (DTD), which is a more suitable representation technique for the Internet. Then, this DTD can be used to represent every conceptual schema of a website as an XML document. Next web tools may be developed to explore the knowledge available in this conceptual schema (XML) document, e.g. to give a visual representation of the CS.
Note that for the transformation of the BR-version of our meta conceptual schema into an XML DTD we have developed a general purpose algorithm to translate BR-schemas into XML document type definitions.

2. Structure and Overview of a Website

2.1. Hierarchical Structure in a Website

Today's website structure is mostly presented in a hierarchical way. It is built up out of pages where one frame (mostly left of the website) is used to depict the menu of subjects (connected with links) accessible from that page. In the other pages information can be found together with some hyperlinks. For a visitor, only the menu gives some minor information about the structure of the website. When a website contains information built up in different layers this menu can only bring information about one layer. Using this technique, it is important to structure your menu's well, making sure that each description used as a menu item is a good representation of what information can be found underneath it.

2.2. Some Techniques to reveal Structure

In this section we will describe briefly two existing techniques to reveal the structure of a website. One to provide an overview of a website to users called sitemaps, and one called structured maps [DM 96], an additional modeling construct superimposed over available information sources, that provides structured and managed access to data. We touch these techniques lightly because they depict in some way what this thesis is all about.

2.2.1. Sitemaps

Sitemaps is a Java application that visualizes a given website (or collection of links). Through a WebCrawler, Sitemaps first traverses every link of the website, collects statistical data, and indexes all the words and pages of the site. Based on the statistical and the indexing, sitemaps converts each page of the site into a vector, and uses these vectors to train a neural network. As the outcome, the trained neural network presents the site in an organized map: subject areas are identified and labeled; their sizes and locations are determined by relationships among the subjects and their occurrence and co-occurrence frequencies. Links are clustered and located within their respective subject areas, represented by colored dots. Some patterns of the site are clearly revealed.
To help users interact with the map, Sitemaps provides various interactive tools. For example, areas can be labeled in more/less details through adjusting a scroll bar; links can be selected through clicking or dragging; contents of any selected links can be shown in a separate window, etc.

Examples of sitemaps can be found :

The Boeing Company
http://lislin.gws.uky.edu/Sitemap/sm2/boeing.html
Dynamic Diagrams, In
http://lislin.gws.uky.edu/Sitemap/sm2/dd.html

An other Java applet of the same kind can be found at

http://www.objectvisionltd.com/java/index.html

Visual Sitemaps displays a directory map of a Website with active links. Visual SiteMap provides an alternative method for navigating sites pages. This applet builds a navigation tree and correlates tree nodes to URLs by reading an ASCII configuration file.

These are two tools that can help users to expose the structure of a website they want to explore looking for certain information.

2.2.2. Structured Maps

Structured maps are based on Topic Navigation maps, defined by the SGML community to provide multi document indices and glossaries. A structured map provides a layer of typed entities and relationships where the entities can have typed references to information elements in the information universe. Structured maps can be placed over loosely structured data, e.g., document collections, with references at various levels of granularity. Structured maps directly support new, customized, and even personalized use of the information.
Structured maps provide the capabilities to organize access to such divers information in an electronic environment and are superimposed over an underlying universe of documentation.

Structured maps can model typed entities and relationships, e.g., Students and Courses. Instances of the entity types are connected to elements (often fragments of some larger source) in the underlying universe of information. Consider the structured map shown in Fig 2.1. Three information sources are shown in the information universe: the student register containing general information of all the students who are a member of the university, the list of student points, and the schedule when the courses are given. This information universe is supplemented with a structured map definition, shown at the top of the figure and a structured map instance, shown in the middle of the figure. In the structured map definition two entity types are introduced, with a relationship type to indicate that students follow courses. OMT [WK 96] notation is used because of the strong similarity of structured maps and entity relationship-style model (ERM).
The student entity has two facet types shown extending from the bottom of the entity symbol. The middle box shows the current instance of the structured map. Each facet instance, for an entity instance, consists of a set of zero or more addresses where each address references an information element from the universe. It is shown in the figure as an envelope icon. This structured map can be used as a navigational guide for the underlying information universe.

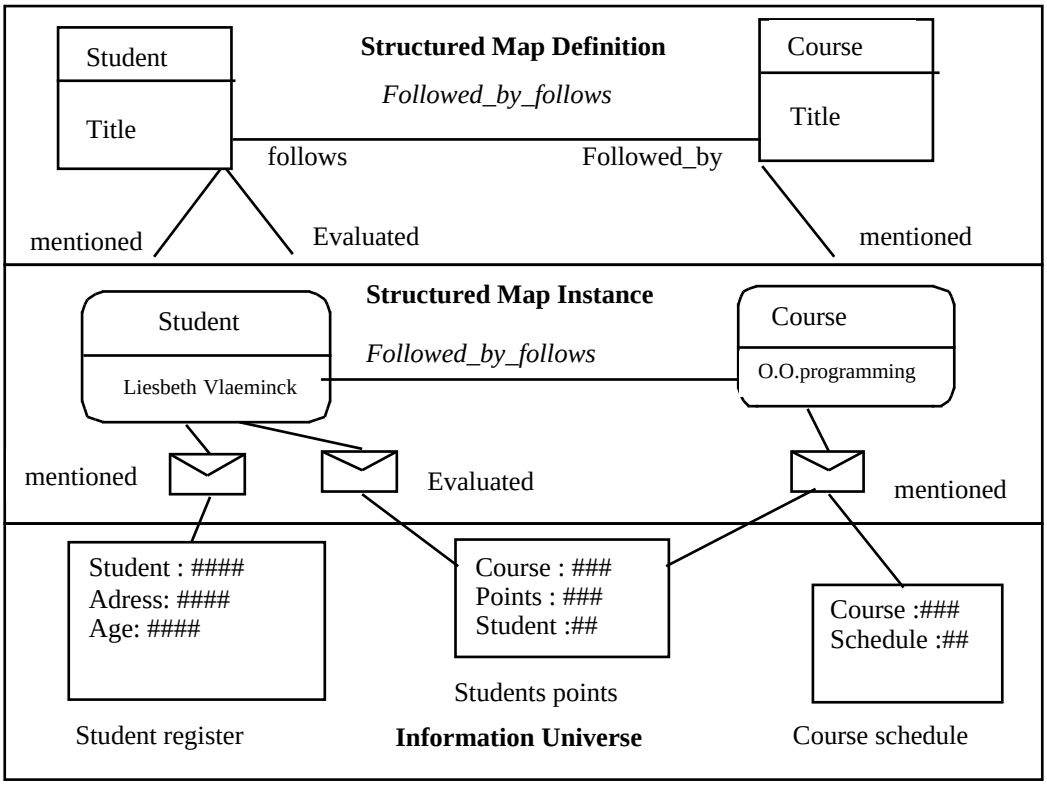

Fig. 2.1 : Example structured map.

As we've seen in the example, structured maps have a three level model, the structured map definition, the structured map instance, and the underlying universe of information with various information elements highlighted by the facets of the structured map.
The modeling of structured maps is fairly elementary compared to most ERD models, but they are currently guided by the definition of the Topic Navigation Map. A Topic Navigation Map is represented as an SGML document and uses the terms: topic, topic relation, topic title, and anchor role as the analogous terms for entity, relationship, title, and facet in structured maps.
The motivations for following Topic navigation maps is :

- The fact that they are being proposed as an ISO standard to provide multi document indices, glossaries and table of contents
- They use a basic entity relationship model at the core.
- Topic Navigation Maps are defined using SGML.
- Topic Navigation Maps use a DTD to describe the structure of the document instance.

A structured map introduces useful information to serve as a structured guide to selected information elements in the underlying universe. When we ignore the difference with web-based approaches to information access, because structured maps are not limited to the structure of any particular web, structured maps are somewhat like structured bookmarks on the World Wide Web.
When we consider a website as being the underlying universe of information, then there are some similarities between structured maps and the information we want to present of websites in this thesis. As earlier mentioned, having a good map or schema

of the universe of information you want to model or use, is of big importance, not only it will help you to design and create the website, it is also an interesting instrument to provide the users of the website with some survey and help them to work in an efficient way with the information.
We will use in this thesis Object Role Modeling (ORM) [HA 95] as a semantic modeling approach which views the world as objects playing roles. Many of its features are derived from NIAM [HA 95] (Natural-language Information Analysis Method)
The conceptual schema of the website gives the type of entities (or objects) the website contains and the relationships between these object types
By creating XML documents of the conceptual model in a common way we will be able to present that structural information to users in a comparable way like the structured maps.
In chapter 7 we will show how we can use JavaScript [NN 96] and the DOM (Document Object Model) [BD 98] to work with the information in the XML documents. We will develop a small example to show how we can work with the received information in the XML documents and how we can present this in a structured way.

3. SGML, HTML en XML

3.1. SGML

SGML (the Standard Generalized Markup Language) is a platform-neutral standard for creating documents and information archives. It is a series of rules that everyone can follow in order to make their documents publishable in different media (print, CD-ROM, the Web) and to make their documents readable with different kinds of computers. SGML is also a structure for storing information that eases info-management and manipulation: it allows very powerful searching, and allows large information repositories to be re-purposed, broken down, and rearranged intelligently into individual documents.

3.1.1. Text Formatters and Formatting Markup

It all started with text processing systems. Text processing is the sub discipline of computer science dedicated to creating computer systems that can automate parts of the document creation and publishing process. The first wave of automated text processing was computer typesetting. Authors would type in a document and describe how they would like it to be formatted. A rendition is what we call the file format that contained the mix of the actual data of the document, plus the description of the desired format. Some well known rendition notation include *Troff, Rich Text Format*, *LaTeX*. The typesetting systems sped up the process of publishing documents and evolved into what we now know as desktop publishing. The user interface to the rendition (the file with formatting code in it) is designed to look like the presentation (the finished paper product). We call this: what you see is what you get (WYSIWYG) publishing. The form of typesetting notation that predates WYSIWYG (and is still in use today) is called *formatting markup.* Depending on the particular formatting markup language the text is circled with instructions called *tags* or *codes.*
SGML is a language where markup is not used for formatting purposes but to describe the structure of the text to make it possible to process the text and the recorded information. This type of markup is called *generalized markup*.
With SGML, organizations in government, aerospace, airlines, automotive, electronics, computers, and publishing have freed their documents from hostage relationships to processing software. SGML coexists with graphics, multimedia, and other data standards needed for Open Information Management (OIM) and acts as the framework that relates objects in the other formats to one another and to SGML documents.

To summarize, it is possible to classify markup as one of three types: stylistic, structural or semantic.

Stylistic Markup
This indicates how the document is to be presented. When we use bolding or italics on a word processor, it is stylistic markup. In HTML <FONT>, <I>, <B>, <U> tags are all stylistic markup.

Structural Markup
This informs us of how the document is to be structured. The <Hn>, <P>, and the <DIV>tags are examples of structural markup, which indicate a heading, paragraph and container section respectively.

Semantic Markup
This tells us something about the content of the data. As such <TITLE> and <CODE> are examples of semantic markup in HTML.

3.2. HTML

HTML as well as XML are based on SGML. HTML, the HyperText Markup Language, is a particular, though very general application of SGML. There is a limited set of markup tags that can be used with HTML.

3.2.1. HTML and the Web

In 1989 a researcher named Tim Berners-Lee proposed that information could be shared within the CERN European Nuclear Research Facility using hyper linked text documents. He was advised to use an SGML-ish syntax by a colleague named Anders Berglund, an early adopter of the new SGML standard. They started from a simple example document type in the SGML standard and developed a hypertext version called the Hypertext Markup Language (HTML).

Relative to the 20 year evolution of SGML, HTML was developed in a hurry, but it did the job. When it came on the scene it sparked a publishing phenomenon. The simplicity of HTML and the other web specifications allowed programmers around the world to quickly build systems and tools to work with the web. Its simplicity is widely believed to be an important part of its success.

HTML inherited some important strength from SGML. With a few exceptions, its element types were generalized and descriptive, not formatting constructs as in languages like TeX and Microsoft Word. This meant that HTML documents could be displayed on text screens, under graphical user interfaces, and even projected through speakers for the sight impaired.
HTML documents used SGML's simple angle bracket convention for markup. That meant that authors could create HTML documents in almost any text editor or word processor. The documents are also compatible with almost every computer system in existence.
On the other hand HTML only uses a fixed set of element types. It is not extensible and therefore can not be tailored for particular document types, and it was not very rigorously defined until years after its invention. By the time HTML was given a formal DTD (Document Type Definition), there were already thousand of web pages with erroneous HTML.

3.2.2. HTML gets extended unofficially

As the web grew in popularity many people started to chafe under HTML's fixed document type. Browser vendors saw an opportunity to gain market share by making incompatible extensions to HTML. Most of the extensions were formatting commands and thus damaged the Web's interoperability. The first golden rule, standardization, was in serious danger.

One argument for implementing formatting constructs instead of (structure) abstractions is that there are a fixed number of formatting constructs in wide use, but an ever growing number of abstractions. Let's say that next year biologists invent a new formatting notation for discussing a particular type of DNA. They might use italics to represent one kind of DNA construct and bold to represent another. In other words, as new abstractions are invented we usually use existing formatting features to represent them. We have been doing this for thousands of years, and prior to computerization, it was essentially the only way.
We human readers can read a textual description of the meanings of the features and we can differentiate them from others using our reasoning and understanding of the text. But this system leaves computers more or less out of the loop.
For instance superscript can be used for trademarks, footnotes and various mathematical constructs. Italics can be used for references to book titles, for emphasis and to represent foreign languages. Without generalized markup to differentiate, computers cannot do anything useful with that information. It would be impossible for them to translate foreign languages, convert emphasis to a louder voice of text to speech conversion, or do calculations on the mathematical formulae.

3.2.3. The World Wide Web reacts

As the interoperability and scalability of the web became more and more endangered by proprietary formatting markup, the World Wide Web (WWW) Consortium (W3C) decided to act. They decided to adopt the SGML convention for attaching formatting to documents, the style sheet.
They invented a simple HTML-specific style sheet language called *Cascading Style Sheets* (CSS) that allowed people to attach formatting to HTML documents without filling the HTML itself with proprietary, rendition-oriented markup. They also came up with a simple mechanism for adding abstraction to HTML. It allowed new abstractions to be invented but provided no mechanism for constraining their occurrence. In other words it brought HTML back to being a single standard, more or less equally supported by the major vendors, and it allowed people to define arbitrary extensions (with many limitations).
But they knew that their stool would not stand long on two of its three legs. The (weakly) extensible HTML and CSS are only stopgaps. For the Web to move to a new level, it had to incorporate the third of SGML's important ideas, that document types should be formally defined so that documents can be checked for validity against them.

Therefore the World Wide Web Consortium decided to develop a subset of SGML that would retain SGML's major virtues but also embrace the Web ethic of minimalist simplicity. They decided to give the new language the catchy name eXtensible Markup Language (XML).

3.3. XML

3.3.1. What is XML?

XML, the eXtensible Markup Language is as we discussed before a streamlined subset of SGML. XML facilities that, like in full SGML, you can use any set of tags. You can literally create your own markup language with XML.

XML was originally conceived as a big brother to HTML. As its name implies, XML can be used to extend HTML or even to define a whole new language completely unlike HTML.

We will now introduce the ideas and techniques that define the language XML to clarify the opportunities of the language and the advantages. In chapter 5 we will use these techniques to transform our meta conceptual schema for websites into an XML document. Only a brief introduction will be given concerning these technique, just enough to understand the theory we describe in chapter 5. For more detailed information we refer to specialized lecture [GP 98] [BD 98].

3.3.2. XML Documents

XML got the name eXtensible Markup Language because it is not a fixed format like HTML. While HTML is limited to a fixed set of tags that the author can use, XML users can create their own tags (or use tags created by others), which actually describe their content. As such it is a meta language.

Consider following example:

```
<Student studentid=”2554123”>
      <Name>Tom Decruyenaere</Name>
      <Age>29</Age>
      <Address>Kortrijksesteenweg 501, 9000 Gent</Address>
      <Discipline>Applied Information Technology</Discipline>
<Student>
```

Here you can see that the tags contain information about their contents. The <Student> tag tells you that the element’s content relate to the student, where <Name> is the tag that contains the name of that student.
Even though you have never seen this example before, you are able to understand its content.

First we will discuss the different parts of an XML document.

The XML Prolog

The prolog is made up out of an XML declaration and a document type declaration, both optional.

The XML declaration is placed at the very beginning of the document file. It indicates the version of XML used to construct the document so that an appropriate parser, or parsing process, can be matched to the document.

```
<?xml version=”1.0”?>
```

The document type declaration declares the document type that is in use in the document. The DTD is a formalization of the intuitive idea of a document type. The DTD lists the element types available and can put constraints on the occurrence and the content of elements and other details of the document structure.

Elements

As we have already seen, the XML document essentially consists of data marked up using tags. Each start-tag / end-tag pair, with the data that may lie between them, constitutes an element.

```
<Name> Tom Decruyenaere </Name>
```

The names in the start and the end tag must be the same. Since XML is case sensitive, care must be taken to ensure that the names match in letter case. XML allows you to give the tags any name you desire. However individual parsing programs may set limits on the length of names even though the XML specification makes no mention of such.

This freedom in the naming of tags is where the power of XML lies. You can give your tags names that describe the content of the element.

To have a well-formed document nearly everything we need to know about how to markup correctly is contained into three simple rules:

- The document must contain one or more elements.
- It must contain a uniquely named element, no part of which appears in the content of any other element, known as the root element.
- All other elements within the root element must be correctly nested.

Attributes

In addition to content, elements may have attributes. Attributes are a way of attaching characteristics or properties to elements of a document. Attributes have names and values. These values are passed to the application by the XML parser but which do not constitutes part of the content of the element.
Attributes are given as part of the elements start tag.

```
<Student StudentID=”2554123”>
```

In the above, ***StudentID*** is an attribute of the ***Student*** tag.

Entities

We will deal with entities shortly because we will not use this property further on in this thesis.

Entities are usually used within a document as a way of avoiding having to type out long pieces of text many times within a document. It provides a mechanism whereby you can associate a name with the long piece of text, and then, wherever you need to place that text within the document you just mention the name instead. This in turn entails that if any modification has to be made to the replacement text, that you only have to make one set of changes to one piece of text, rather than having to make a large number of alterations within the document.

3.3.3. Document Type Definition (DTD).

The DTD defines the allowed element types, attributes and entities and can express some constraints on their combination. A document that conforms to its DTD is said to be valid.
A DTD consist of the declaration within the DOCTYPE declaration, which includes entity, element and attribute declarations, and any other declaration required for describing our documents. XML provides a mechanism to refer to an external DTD file using the following syntax:

```
<!DOCTYPE  University  SYSTEM “university.dtd”>
```

This allows us to reference a DTD stored in a separate file and avoids having to copy the same DTD into every document we are working with. We can also replace the file name by an URI (Uniform Resource Identifiers) address.
This way the DTD can be built up out of an internal and an external subset. We can place declarations in both.

```
<!DOCTYPE University  SYSTEM “university.dtd”
[
<!ELEMENT Object ( element1 | element2)>
]>
```

Element Declaration

Elements are the foundation of XML markup. Every element in a valid XML document must conform to an element type declared in the DTD.

```
<!ELEMENT Student ( Name , Age , Address , Discipline)>
```

Element type declarations have a content specification. In the example above the element ***Student*** must contain a ***Name***, ***Age***, ***Address***, and ***Discipline*** element.

Element Content Model

Each element is described within the declaration by a content model that lists the subsidiary elements it can contain, with some elements defined as being optional or repeated. The content model is part of the element declaration and is enclosed in the parenthesis. The model describes what lower level elements make up the current element. There are no restrictions upon what you name these subsidiary elements provided that you follow the rules for naming.

The symbols used for XML content models are the same as a subset of those used for regular expressions. For example, parentheses are used for grouping; a question mark is used to signal optional elements; asterisk and plus symbols are used to signal repetition. We also find symbols to denote the sequencing of elements. The use of the comma symbol in a content model means that the content elements must occur in that order. The | symbol is used to indicate a selection. An overview is given in Table3.1

Symbol	Usage
,	Strict ordering
|	Selection
+	Required and repeatable
	Minimum 1
*	Optional and repeatable
?	Optional
( )	Grouping

Table 3.1 : Content model operators and their usage.

#PCDATA

Eventually we reach an element that will have character content. This can be indicated in the content model of that element by including the special element name #PCDATA. This element can also be used in combination with others.

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

EMPTY

Sometimes we want an element that can never have any content. We need the element name to act as a placeholder in the document structure but we do not want the XML parser to interpret the content itself

```
<!ELEMENT succesfull EMPTY >
```

ANY

This lets us put any type of content within the element. The consensus says that it is only useful as a temporary measure when developing a document type definition.

Attribute List Declaration

An element can have attributes associated with it. Attribute declarations contain the element name to which the attribute belong; the names of the attributes in the element; the data types or possible values for the attributes contents; and default values for the attributes.

```
<!ATTLIST Student StudentID ID #REQUIRED >
```

An element can have any number of attributes associated with it. With DTDs we can control how the attribute value is interpreted.

In the table 3.2, on the next page, all the possible content data types for an attribute are listed.

Attribute Type	Attribute value content
CDATA	Character data
ID	A name that is unique over all other ID attribute values
IDREF	A name that is defined by some other ID type attribute
IDREFS	A series of names that are defined by ID type attributes and are separated by white space
ENTITY	A name of a pre-existing external entity. The entity is assumed to contain binary data
ENTITIES	A series of names of pre-existing external entities that are separated by white space. The entities are assumed to contain binary data.
NMTOKEN	A name, which cannot begin with the character _ or :
NMTOKENS	A series of NMTOKEN values separated by white space
NOTATION	A series of NMTOKEN values separated by white space that have been named in NOTATION declaration
Enumeration	A series of NMTOKEN values that you explicitly listed in the attribute declaration

Table 3.2 : Summary of attribute types.

Default values of attributes

Attributes can have default values. If the author does not specify an attribute value then the processor supplies the default value if it exists. Specifying the default can be done by including it after the type or list of allowed values in the attribute list declaration.

Furthermore XML allows us to have some control over the presence of values for the attributes by the following words: #REQUIRED , #FIXED, #IMPLIED.

The #REQUIRED flag forces the attribute to appear in each start-tag for this element. That is, there is no default value for any attribute with this mark against it. Whereas the #IMPLIED flag indicates that the application supplies the default value if one isn't supplied in the document instance. Any attribute that has #FIXED attached to it means that it can only have the one value when the attribute is used in that element.

4. The Conceptual Schema of a Website

What we are seeking for is a means to describe the content and the structure of a website in a general and formal way, such that it can be used by browsers to display the structure of the website and allow users to see at a glance what kind of information the website is offering. Also search engines should be able to explore this information.

A common way in databases to describe the type of information they are dealing with and the way it is structured is a conceptual schema. The conceptual schema of a database gives the type of entities (or objects) the database contains and the relationships between these object types. As an example, consider a university database. Typical object types are Student, Course, and Lecture. The relationships between these object types are e.g. “Student follows a Course” and “Lecture gives a Course”. The conceptual schema of a database further describes the properties of the object types (called attributes) and the constraints that apply to attributes and relationships.

When we consider a website as being a forum to deliver information, then there are some similarities with a database. Therefore we may consider to use the technique of conceptual schemas from database theory to describe the content and structure of a website. However a website is not a database. A website may contain structured data, but it may also contain unstructured data. In addition the organization of a website is different from that of a typical database. A website is organized into pages and links between pages and pieces of information. This means that the basic building blocks of a conceptual schema for a database, being object types, relationships and constraints, are not necessarily suitable and certainly not sufficient to build the conceptual schema of a website. Therefore, we first have to establish the basic concepts needed to specify the conceptual schema for a website. We have to find out what exactly of a website we want to describe with a conceptual schema. E.g. do we merely describe the (type of) information in the website, or do we describe the page and link structure of the website or both? In addition, there is the question of what level of abstraction we are looking for? Because the distinction between instance and type is not as clear as in a database, this may be a difficult issue. We give an answer to these questions in section 4.1.

In database theory, the set of basic concepts used to describe a conceptual schema is usually described by the same technique, i.e. a conceptual schema. Because this is a conceptual schema of conceptual schemas, this schema is called the *Meta Conceptual Schema*. Therefore our first job will be to develop the meta conceptual schema for a website. This is done in section 4.2. To represent the meta conceptual schema we have used the Binary Role (BR) Modeling approach, which views the world as objects playing roles. The basic building blocks of the BR Model are Object Types (OT) (graphically represented as circles) and binary relationships composed of two roles (graphically represented as a rectangle composed into two boxes, each box connects with a line to the corresponding OT).

4.1. The Objectives of a Conceptual Website Schema

In database theory, the conceptual schema concentrates on the conceptual issues of the allowable contents and the meaning of the database. Implementation issues such as the grouping into records and the available access paths are not described in the conceptual schema. These are part of the database schema. If we would follow this philosophy for websites, it would only be possible to derive from the conceptual schema what kind of information is available in the website, but we would not be able to derive where it can be found in the website. For databases this is not a problem because a database is used through an application program that hides the database structure from the user. For a website the only existing application is the webbrowser, which does not hide the pages and links from the user. On the contrary, pages and links are the basic manipulation concepts to be used in a webbrowser. Therefore to fulfill our original goal, to allow users to see at a glance where and what kind of information can be found in a website, the conceptual schema of a website should also describe where the information can be found in the website. To realize this a conceptual schema for a website will be composed of three parts: one part will describe what kind of information is available in the website (called the *content schema*), a second part will describe the page and link structure of the website (called the *structure schema*) and the third part will describe where the information can be found in the website. This last part connects the information in the content schema to the information in the structure schema. Therefore this part is called the *connection schema*. We have opted to clearly identify those different parts rather than intertwine all the information into one schema. This will offer more flexibility towards the future as web technology changes quickly.
In the next section, we describe each of these sub-schemas into more detail by means of the meta conceptual schema.

4.2. The Meta Conceptual Schema

As already explained earlier, we will use a meta conceptual schema to define the concepts of a conceptual schema for a website.
To define a conceptual schema of a website you need certain concepts which can be used to represent the different information components present in the website and build them together according to a certain structure. The meta conceptual schema will be the frame in which the presentation of these concepts and their relation with each other will be determined.
As a result it should be possible to represent a conceptual schema of any website using the declared concepts of the meta conceptual model. This conceptual schema should indicate what information can be found in the website and how this information is linked together.
As a next step in our theory we will use XML to build a Document Type Definition (DTD) containing the declarations of the different concepts defined in the meta conceptual schema. This makes it as an information system more robust by forcing the conceptual schemas that we will create of websites using this DTD to be consistent (see Chapter 5).

As already explained the meta conceptual schema will be divided into three parts:

- The Content schema.
- The Structure schema.
- The Connection schema.

The Content schema.
In this schema we define the different conceptual ‘elements’ which represent the content of a website and their (conceptual) relation to each other. To do this, we make a distinction between Concepts and Concept Types. As an example to clarify the difference between these two elements we can use "Student" that is a Concept Type whereas "the student Tom Decruyenaere" can be considered as a Concept. Like in ORM Objects play roles creating relations between each other. Object and relations are both subtypes of Concepts, Object Types and Relation Types are subtypes of Concept Types. Objects may be structured, constructed out of other Objects or just simply elementary. When we define a tree as being an Object constructed out of branches and leafs it can be considered as a structured object whereas a leaf can be an elementary object. We will ignore the structured Objects further on in this thesis to keep it from becoming to complex. Relations exist out of Roles. The Object Type Student has a Relation Type with an Object Type Course because the first plays a role on the second. A Student follows a Course. When we keep it general like in this example, a role between two Object Types creates a Relation Type. When we become specific and indicate that the Student "Tom" follows a Course " Software Engineering" we talk about a Relation between two Objects.
Fig. 4.1 gives a graphical representation of the content schema.

The Structure schema.
In this schema we define the ‘elements’ who are typical for building a website and defining the structure of it: pages and links. We also define the concept of page prototype and link prototype. These concepts allow to define a general structure for pages and links. This makes it possible to define a page or a link as an ‘instance’ of these prototypes, which means that they must be conform the prototype. E.g. Each Student could have a website page where some specific information concerning this student is gathered. The page for this particular student can be considered as a page and as an instance of the Page Prototype defining structure of this Page. This does not mean that every page or link should have a prototype.
Fig. 4.2 gives a graphical representation of the structure schema.

The Connection schema.
This schema allows to connect the first two schemas. It provides us with the possibility to indicate how the conceptual content of a website is reflected on the structure. More specific, this schema defines the relation indicating what concept (type)s can be found on what page (prototype)s. The OT Concept Type is connected to the OT Page through the relation created by the two roles ***with_overview_on*** and ***overview_page_of***.
Fig. 4.3 gives a graphical representation of the connection schema.

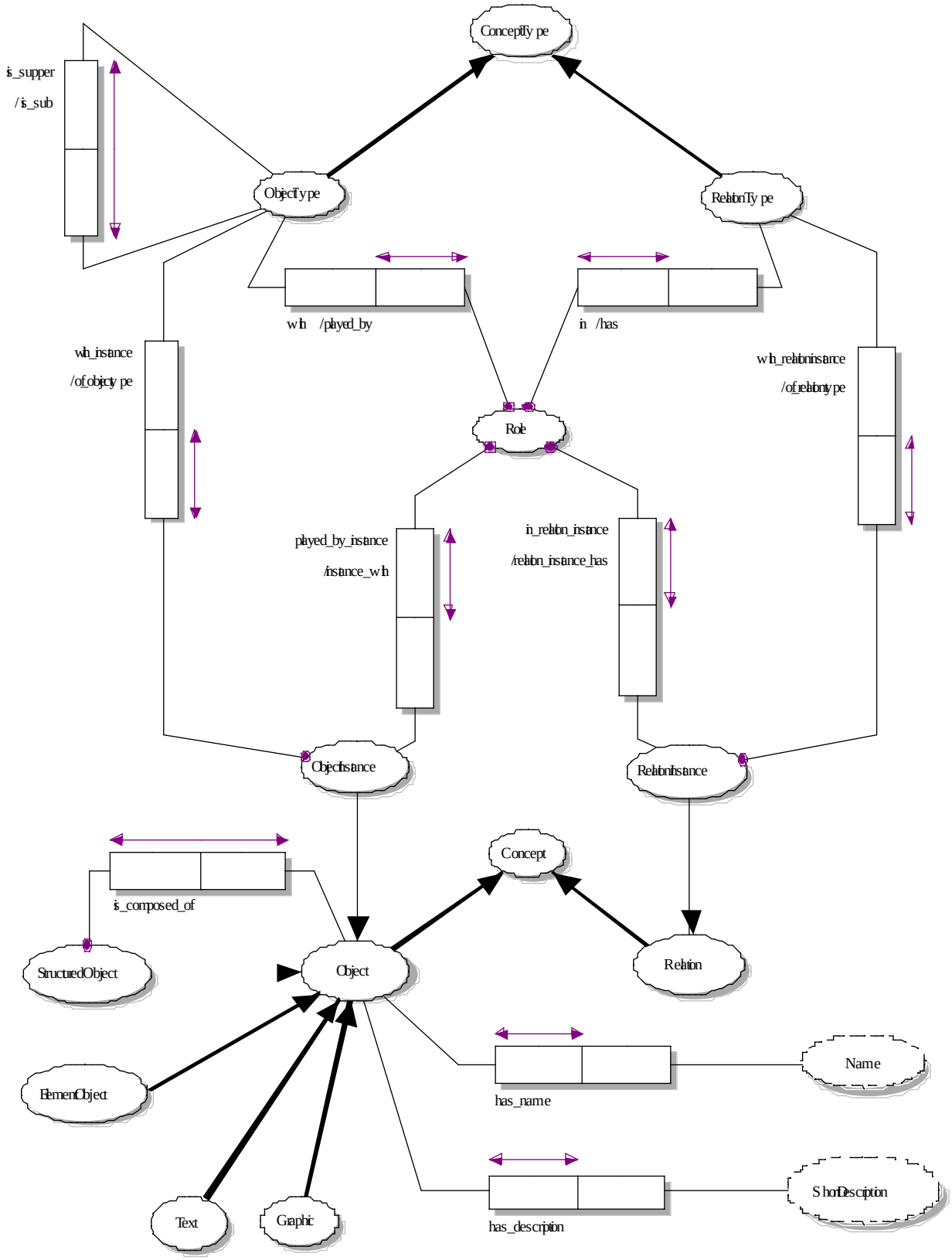

Fig. 4.1 : Content schema of website.

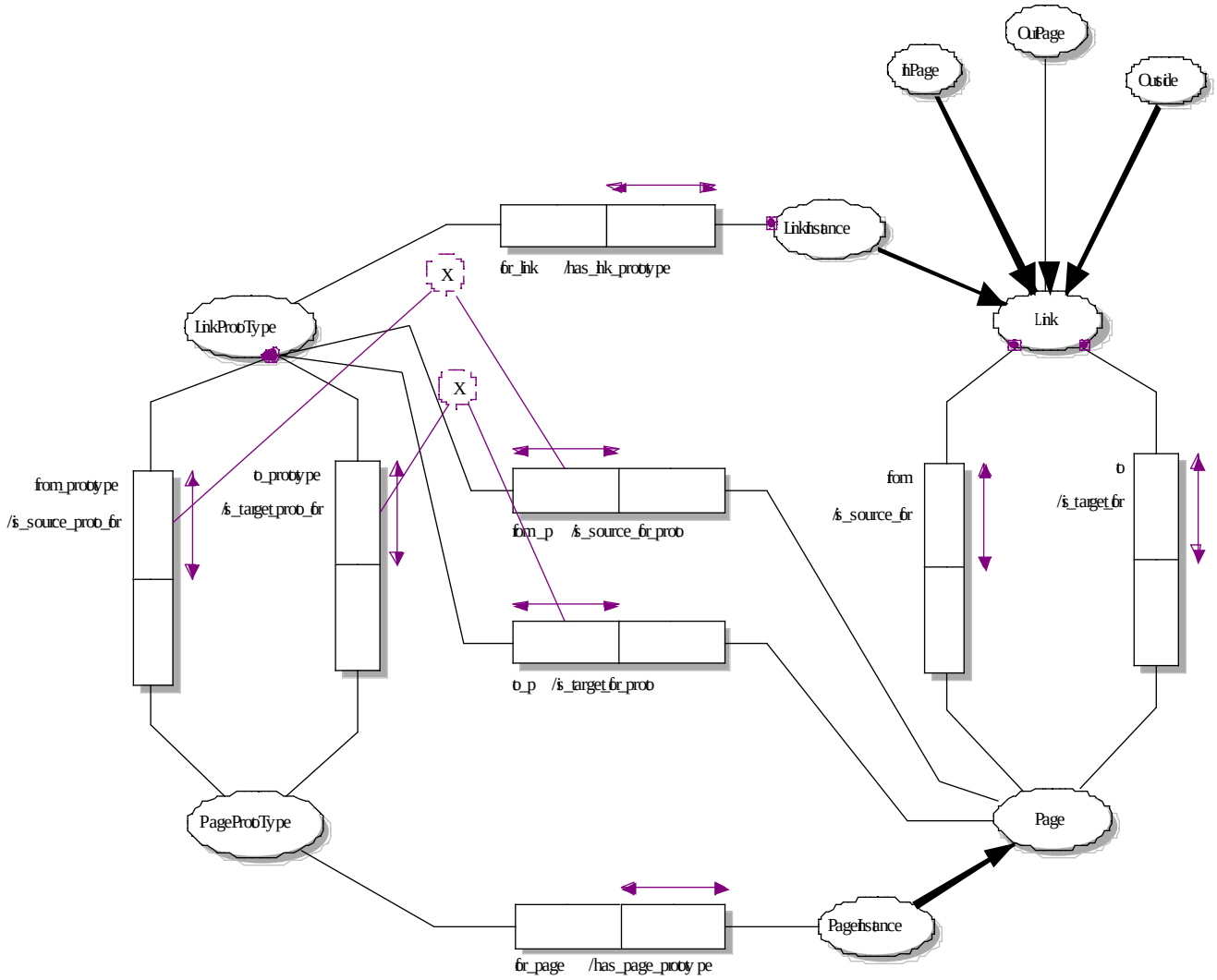

Fig. 4.2 : Structure schema of a website.

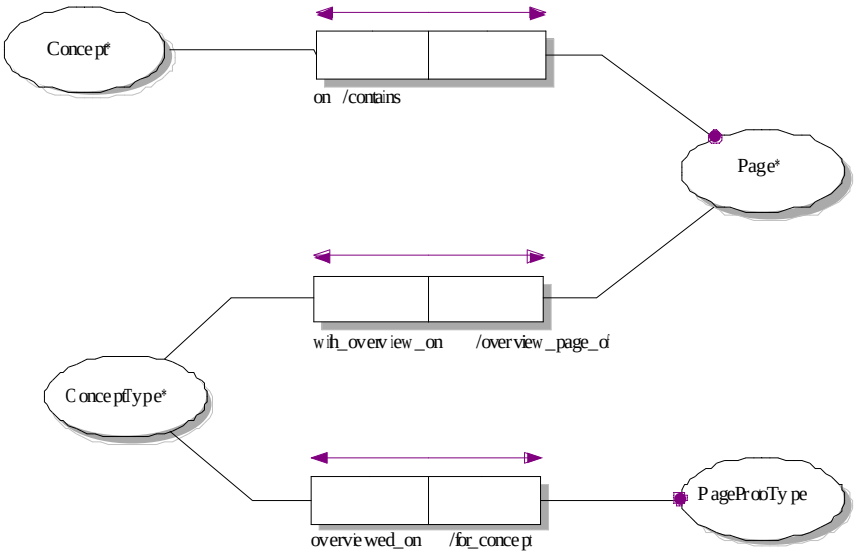

Fig. 4.3 : Connection schema of a website.

4.3. Name Convention.

All the different OTs in the conceptual schema are written with a capital letter without spaces between different words. If words are constructed out of more words we also use capital letters to make it more readable.

For the relations and more specific the roles, we use small letters and underscores to connect the different words in a role.

In traditional conceptual schemas it is allowed to use the same role names in different relations. As an example “has” is a frequently used role name. However, we will see in section 5.1.1 that the algorithm defined to transform a conceptual schema into a XML DTD requires unique names for all roles. Therefore we should take care that every role has a unique name. As an example, in the meta conceptual schema we use for the OT ***Link*** and the OT ***LinkProtoType*** different role names: ***from*** and ***from_prototype***. Normally it should be more convenient if we use two times ***from*** but it will become clear when we explain the algorithm to transform a conceptual schema into a DTD in XML why this would induce problems.

5. Using XML to represent a Conceptual Website Schema

The BR Model is not a very convenient formalism to be used on the Internet. The Internet is nowadays dominated by HTML, but the language of the future seems to be XML. As explained in chapter 3, XML allows to bring more structure and semantics to the web. Therefore a suitable representation for a conceptual website schema could be XML. What we are aiming for is a standard way to represent the conceptual schema of a website on the web. This can be accomplish by defining a DTD for describing a conceptual website schema. Each conceptual website schema can then be expressed as an XML document using this DTD. Because this DTD will describe the general structure of any conceptual schema for a website it is in fact the same as the meta conceptual schema defined in the previous chapter. What we need to do is to translate the meta conceptual schema given in BR formalism into a XML DTD.
To transform the meta conceptual schema into a XML DTD we have developed an algorithm that is capable to transform any BR-schema into a DTD. This algorithm is given in section 5.1 and illustrated in a simple example. In section 5.2 the algorithm is applied on the meta conceptual schema resulting in the requested DTD, which we will refer to as the *Conceptual Schema DTD*.
We will also use a simplified example to show how we can present whatever conceptual schema in XML using this conceptual schema. Finally we will test this theory in a Case-study (Chapter6).

5.1. Algorithm to transfer a BR Conceptual Schema into an XML DTD

We will explain this algorithm on the basis of a very simple example. We will use a conceptual schema of an academic domain were we have some OTs and relations between them. We assume in this domain that a course is given by only one professor and that a student is obliged to follow a course. Both students and professors are members of the University.

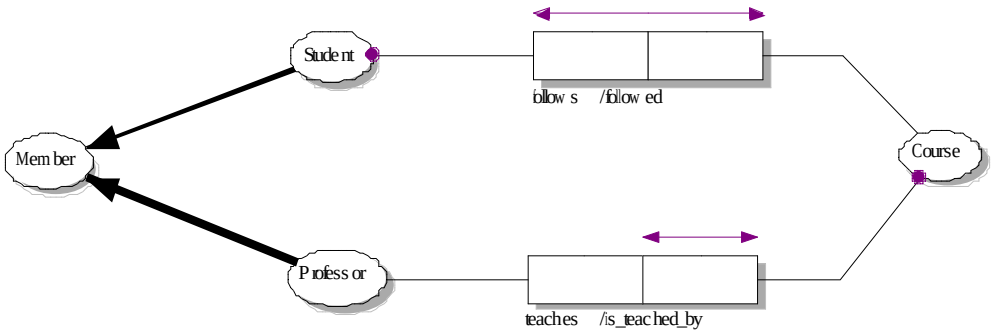

Fig. 5.1 : Simplified conceptual schema of Academic Domain.

We use the theory of DTD given in chapter 3 to transform this conceptual schema into a DTD. As the first transformation rule we can state:

Every OT in the schema is declared in the DTD as an Element.

When we look at the OT ***Member*** we see in the schema that it has two subtypes. We will now explain how we represent these subtypes within the element content model of the element ***Member***.

```
<!ELEMENT Member(Student | Professor)>
```

The element Member has two subsidiary elements declared in its content model. These are the subtypes in the conceptual schema. This brings us to the second transfer rule:

To represent a subtype relationship we specify the element for the subtype in the declaration of the content model of the element for the supertype.

E.g. The element ***Member*** is either an element ***Student*** or an element ***Professor***.

Note that within a group (….) the connector symbols must be the same. We cannot mix them like , (element1 | element2 , element3) unless we introduce more parenthesis, ((element1 | element2), element3). Unlike arithmetic expressions there is no implied precedence between the connectors.

For each element we define an ID as an attribute. The value for this ID will be used to refer to this element. This is necessary to create relations between elements. Relations are no more then roles played by one element sometimes on another element.

```
<!ATTLIST Member MemberID ID #REQUIRED>
```

To translate roles we make use of the following rule:

We declare a role as an element. When the role is played on a NOLOT (NOn Lexical Object Type) it is declared as an EMPTY element with an IDREF in the attribute list declaration. When it is played on a LOT the content is #PCDATA.
The role creates the relation between two elements by use of the ID reference.

Because every role is transformed into an element in the XML DTD by this rule, we have to make sure that we use for every role in the conceptual schema a different name.

The name of the role is part of the element type content model of the element representing the OT that plays the role. When an OT has different roles they will be lined up in a sequence in the content model of the element. The comma between the different roles implements this. We use the occurrence indicator to make roles optional.

```
<!ELEMENT Course ( is_teached_by , is_followed*) >
<!ATTLIST Course  CourseID ID #REQUIRED >
      <!ELEMENT is_teached_by EMPTY>
      <!ATTLIST is_teached_by ProfessorID ID #REQUIRED>
      <!ELEMENT is_followed EMPTY>
      <!ATTLIST is_followed StudentID ID #REQUIED>
```

In our example, ***Course*** plays two roles into two relations. The role ***is_teached_by*** presents the relation with a professor who is teaching this course. We indicate with the name of the ID that it should be a professor. However we cannot use constraints within this model to oblige the use of professor ID's, in fact every valid ID value can be used.

If we take the example of ***Student*** and its role ***follows***, we can see that the uniqueness constraints, present on this role, can be transformed using these occurrence indicators. ***Follows*** is part of a many to many relation and is mandatory, that is why we use a required and repeatable operator + . When we take the OT ***Professor*** were ***teaches*** is part of a many to one relation which is not mandatory at the professor side we use the optional and repeatable operator *.

```
<!ELEMENT Student ( follows )+ >
<!ELEMENT Professor ( teaches )* >
<!ELEMENT Course ( is_teached_by , is_followed*) >
```

Uniqueness constraints and the mandatory role constraints can be expressed in the DTD by making use of the content model operators as follows:

Symbol	Usage	Constraint
+	Required and repeatable	Mandatory
	Minimum 1	Mandatory and unique
*	Optional and repeatable	Non mandatory
?	Optional	Non mandatory and unique

Table 5.1 : Expression of constraints by content model operators.

When we apply these rules to the entire example conceptual schema we get the following DTD:

```
<?xml version="1.0"?>
<!DOCTYPE Academic_Domain [

<!ELEMENT Member (Professor | Student )>
<!ATTLIST Member MemberID ID #REQUIRED>
<!ELEMENT Student ( follows )+ >
<!ATTLIST Student StudentID ID #REQUIRED >
            <!ELEMENT follows EMPTY>
            <!ATTLIST follows CourseID ID #REQUIRED>

<!ELEMENT Professor ( teaches )* >
<!ATTLIST Professor     ProfessorID ID #REQUIRED>
            <!ELEMENT teaches EMPTY>
            <!ATTLIST teaches CourseID ID #REQUIRED>

<!ELEMENT course ( is_teached_by , is_followed* )>
<!ATTLIST Course  CourseID ID #REQUIRED >
            <!ELEMENT is_teached_by EMPTY>
            <!ATTLIST is_teached_by ProfessorID ID #REQUIRED>
            <!ELEMENT is_followed EMPTY>
            <!ATTLIST is_followed StudentID ID #REQUIRED>
]>
```

With this algorithm we are now able to transform every BR conceptual schema into a DTD. This DTD is then the frame in which the markup is declared to make XML documents.
In our example of the academic domain we could now make instances of XML pages with information following the conceptual schema of the academic domain.

```
<?xml version="1.0"?>
<!DOCTYPE Academic_Domain SYSTEM “Academic.dtd”>

<Member MemberID=”Member Janssens B.”>
<Professor ProfessorID=”Janssens B.”>
<teaches CourseID=“Object Oriented Programming”>
<teaches CourseID=“Software engineering” >
</Professor>
```

```
</Member>
<Member MemberID=”Member Liesbeth Vlaeminck”>
<Student StudentID=”Liesbeth Vlaeminck”>
<Follows CourseID=“Object Oriented Programming”>
<Follows CourseID“Software engineering” >
</Student>
</Member>
<Member MemberID=”Member Bert Lefever”>
<Student StudentID=”Bert Lefever”>
<follows CourseID“Object Oriented Programming”>
</Student>
</Member>
<Course CourseID=”Object oriented programming”>
<is_teached_by ProfessorID=”Janssens B.”>
<is_followed StudentID=”Liesbeth Vlaeminck”>
<is_followed StudentID=”Bert Lefever”>
</course>
<Course CourseID=”Software engineering”>
<is_teached_by ProfessorID=”Janssens B.”>
<is_followed StudentID=”Liesbeth Vlaeminck”>
</course>
```

5.2. Transformation of the Meta Conceptual Schema of a Website into a DTD

We can now use the rules of the algorithm of section 5.1 to create the conceptual schema DTD of a website. First we give an example of the different applications of rules.

```
      <!ELEMENT   ConceptType   ((ObjectType   |   RelationType   ),
with_overview_on* , overviewed_on*)>
```

ConceptType is an OT and therefore declared as an element in the DTD. In the conceptual schema we see that ***ObjectType*** and ***RelationType*** are sub-entities of ***ConceptType***. This is covered by the second rule in our algorithm and is represented by the two subsidiary elements in the content model of ***ConceptType***. “|” indicates that it is either an ***ObjectType*** or a ***RelationType*** but never both.

```
<!ATTLIST ConceptType   ConceptTypeID ID #REQUIRED >
```

In the attribute declaration of the element ***ConceptType*** is the definition of the ID attribute ***ConceptTypeID***. This attribute will be used to make references in the roles played by other elements on this element (OT in the conceptual schema).

The roles played by the ***ConceptType*** are also present in the content model of the element: ***with_overview_on*** and ***overviewed_on*.** The use of an asterix * reflect the non mandatory and not unique constraints for these roles.

The declaration of the role elements is depict here below.

```
      <!ELEMENT with_overview_on EMPTY>
      <!ATTLIST with_overview_on PageID IDREF #REQUIRED>
      <!ELEMENT overviewed_on EMPTY>
      <!ATTLIST overviewed_on PageProtoTypeID IDREF #REQUIRED>
```

Because the reference of the roles is towards a NOLOT (the role is played on an NOLOT) it is declared as an EMPTY element with an ID attribute.

The complete DTD:

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


<!ELEMENT Website (ConceptType | Concept | Page | PageProtoType |
Role | Link | LinkProtoType)*>
<!ATTLIST Website   WebsiteID ID #REQUIRED >




  <!ELEMENT ConceptType ((ObjectType | RelationType ),
            with_overview_on* , overviewed_on*)>
  <!ATTLIST ConceptType       ConceptTypeID ID #REQUIRED >
            <!ELEMENT with_overview_on EMPTY>
            <!ATTLIST with_overview_on PageID IDREF #REQUIRED>
            <!ELEMENT overviewed_on EMPTY>
            <!ATTLIST overviewed_on PageProtoTypeID IDREF #REQUIRED>


      <!ELEMENT ObjectType ((is_super | is_sub )* ,with*) >
      <!ATTLIST ObjectType    ObjectTypeID ID #REQUIRED >
            <!ELEMENT is_super EMPTY>
            <!ATTLIST is_super ObjectTypeID IDREF #REQUIRED>
            <!ELEMENT is_sub EMPTY>
            <!ATTLIST is_sub ObjectTypeID IDREF #REQUIRED>
            <!ELEMENT with EMPTY>
            <!ATTLIST with RoleTypeID IDREF #REQUIRED>


      <!ELEMENT RelationType ( has+ )>
      <!ATTLIST RelationType  RelationTypeID ID #REQUIRED >
            <!ELEMENT has EMPTY>
            <!ATTLIST has RoleID IDREF #REQUIRED>
```

```
<!ELEMENT Role (played_by , in )>
<!ATTLIST Role  RoleID ID #REQUIRED >
          <!ELEMENT played_by EMPTY>
          <!ATTLIST played_by ObjectTypeID IDREF #REQUIRED>
          <!ELEMENT in EMPTY>
          <!ATTLIST in RelationTypeID IDREF #REQUIRED>


    <!ELEMENT ObjectInstance  ( of_objecttype? ) >
    <!ATTLIST ObjectInstance      ObjectInstanceID ID #REQUIRED >
    <!ELEMENT RelationInstance ( of_relationtype ) >
    <!ATTLIST RelationInstance    RelationInstanceID ID #REQUIRED >

          <!ELEMENT of_objecttype EMPTY>
          <!ATTLIST of_objecttype ObjectTypeID IDREF #REQUIRED>
          <!ELEMENT of_relationtype EMPTY>
          <!ATTLIST of_relationtype RelationTypeID IDREF #REQUIRED>


<!ELEMENT Concept ( ( Object | Relation ),is_on_page? )>
<!ATTLIST Concept     ConceptID ID #REQUIRED >
          <!ELEMENT is_on_page EMPTY>
          <!ATTLIST is_on_page PageID IDREF #REQUIRED>


    <!ELEMENT Object ((StructuredObject? | ElementObject?) ,
          ObjectInstance? , has_name? , has_description? ) >
    <!ATTLIST Object  ObjectID ID #REQUIRED >
          <!ELEMENT has_description (#PCDATA)>
          <!ELEMENT has_name (#PCDATA)>

    <!ELEMENT StructuredObject (is_composed_of*)>
    <!ATTLIST StructuredObject    StructuredObjectID ID #REQUIRED >
          <!ELEMENT is_composed_of EMPTY>
          <!ATTLIST is_composed_of  ObjectID IDREF #REQUIRED>

    <!ELEMENT ElementObject ( Text | Graphics | ObjectInstance)>
    <!ATTLIST ElementObject       ELementObjectID ID #REQUIRED >
          <!ELEMENT Text (#PCDATA)>
          <!ELEMENT Grafics (#PCDATA)>




    <!ELEMENT Relation ( RelationInstance )>
    <!ATTLIST Relation      RelationID ID #REQUIRED >


<!ELEMENT Page (PageInstance?, contains* , overview_page_of* ,
          is_source_for* , is_target_for* , is_source_for_proto*,
          is_target_for_proto* ) >
<!ATTLIST Page        PageID ID #REQUIRED >
          <!ELEMENT contains EMPTY>
          <!ATTLIST contains ConceptID IDREF #REQUIRED>
          <!ELEMENT overview_page_of EMPTY>
          <!ATTLIST overview_page_of ConceptTypeID IDREF #REQUIRED>
          <!ELEMENT is_source_for EMPTY>
          <!ATTLIST is_source_for LinkID IDREF #REQUIRED>
          <!ELEMENT is_target_for EMPTY>
          <!ATTLIST is_target_for LinkID IDREF #REQUIRED>
          <!ELEMENT is_source_for_proto EMPTY>
```

```
          <!ATTLIST is_source_for_proto LinkProtoTypeID IDREF
                                                        #REQUIRED>
          <!ELEMENT is_target_for_proto EMPTY>
          <!ATTLIST is_target_for_proto LinkProtoTypeID IDREF
                                                        #REQUIRED>



    <!ELEMENT PageInstance   (has_page_prototype)>
    <!ATTLIST PageInstance  PageInstanceID ID #REQUIRED >
          <!ELEMENT has_page_prototype EMPTY>
          <!ATTLIST has_page_prototype  PageProtoTypeID IDREF
                                                        #REQUIRED>


<!ELEMENT PageProtoType ( for_concept* , is_source_proto_for* ,
                is_target_proto_for*)>
<!ATTLIST PageProtoType PageProtoTypeID ID #REQUIRED>
          <!ELEMENT for_concept EMPTY>
          <!ATTLIST for_concept ConceptTypeID IDREF #REQUIRED>
          <!ELEMENT is_source_proto_for EMPTY>
          <!ATTLIST is_source_proto_for LinkProtoTypeID IDREF
                                                        #REQUIRED>
          <!ELEMENT is_target_proto_for EMPTY>
          <!ATTLIST is_target_proto_for LinkProtoTypeID IDREF
                                                        #REQUIRED>



<!ELEMENT Link  ( LinkInstance?, from, to )>
<!ATTLIST Link        LinkID ID #REQUIRED >
          <!ELEMENT from EMPTY>
          <!ATTLIST from    PageID IDREF #REQUIRED>
          <!ELEMENT to EMPTY>
          <!ATTLIST to      PageID IDREF #REQUIRED>

    <!ELEMENT LinkInstance   (has_link_prototype )>
    <!ATTLIST LinkInstance  LinkInstanceID ID #REQUIRED>
          <!ELEMENT has_link_prototype EMPTY>
          <!ATTLIST has_link_prototype  LinkProtoTypeID IDREF
                                                        #REQUIRED>

<!ELEMENT LinkProtoType   ((from_prototype | from_p) ,
          (to_prototype | to_p) )>
<!ATTLIST LinkProtoType LinkProtoTypeID ID #REQUIRED >
          <!ELEMENT from_prototype EMPTY>
          <!ATTLIST from_prototype  PageProtoTypeID IDREF
                                                        #REQUIRED>
          <!ELEMENT to_prototype EMPTY>
          <!ATTLIST to_prototype  PageProtoTypeID IDREF #REQUIRED>
          <!ELEMENT from_p EMPTY>
          <!ATTLIST from_p  PageID IDREF #REQUIRED>
          <!ELEMENT to_p EMPTY>
          <!ATTLIST to_p  PageID IDREF #REQUIRED>
```

6. Case Study : Volvo IT Belgium, Y2K Status Information

6.1. Introduction

In this chapter we will model (using the BR-model formalism) the conceptual schema of an Y2K website and the information it contains. We use as a concrete example the website of Volvo Information Technology Belgium representing the Y2K status information necessary to provide a good forum to deal with the questions of the different customers. In Appendix A you can find some more general information regarding this case (The situation sketch).
Next the conceptual schema will be transformed into an XML document using the conceptual schema DTD developed in chapter 5.

6.2. Modeling the Conceptual Schema

As in the modeling of the meta conceptual schema of a website we will also divide the information into three parts rendering the content of the Y2K website, the structure and how these two are connected with each other.

6.2.1. Technical Customer Information Needs

Queries received from the customer vary through the range of all types of infrastructure components. Information is needed in the areas of:

- Class; Servers and clients
- UNIX
- Mid-Range (AS400)
- Mainframe

- Network

There is the concern as to the status of the above-mentioned items from a customer perspective. That means that the solution will also need to follow this approach. Questions arise as to what is being done, whether the specific item has been cleared for Y2K compliance and if not, when it will be. Today there are numerous sources for such information, but from a customer point of view, they are fragmented and platform based, making it hard for the customer to come to a complete overview of what is applicable for him. There is no official point of contact for such information and therefore no control over de messages that are being delivered, their reliability and their form. Customers should be provided with such a forum by VITB and notified of its existence once it is in production.

6.2.2. Kind of Information

6.2.2.1. Technical Information

The items presented below were identified as information needs from the side of the customer and what should be presented for each of them. For each of the items, a subdivision should be made in terms of Hardware, Operating System and Software.

- CLASS
  - Server HW/OS/SW
  - Client HW/OS/SW
- UNIX HW/OS/SW
- Mid-Range (AS400) HW/OS/SW
- Mainframe HW/OS/SW
- Network HW/OS/SW

For each of the items, the following information should be presented:

- Indication of progress, e.g. XX%complete
- Y2K status and dates, possibly using preformatted expressions:
  - not started – specifying planned start date and planned completion date
  - in progress – specifying start date and planned completion date
  - completed – specifying start date and completion date
- Possibility to consult more detailed information, if available
- Contact person for gathering more extensive information. It should be kept in mind that someone should be available at all times. If the contact person is away for a certain period, this must, either be stated, or queries need to be redirected elsewhere.

On a first level, the information should be presented in a clear way, easy to overlook in one glance. On a second level, a report must be built as to the individual status of all e.g. CLASS servers that bear reference to that specific customer, with status information on their individual progress.

6.2.2.2. Meta-information

The presentation of the technical information should be properly introduced and accompanied by a statement that the information is official, reliable and up-to-date and binding. The advice is that the selected medium (see below) is presented as the only official one from a certain point onwards. The customers need to be made aware of this through this medium, and if needed, supported by another depending on the characteristics of the primary one.
There should also be a section on General information leading to relevant information and related links:

6.2.2.3. General Info

- What isY2K?
- What are the generic responsibilities of VITB towards its customers when it comes to Y2K issues?
- What does the Rollover Plan look like?
- Information per platform (currently available info)
- …

6.2.2.4. Links

- Swedish Y2K site
- Aps2 conditions
- …

6.2.3. Conceptual schema of the content

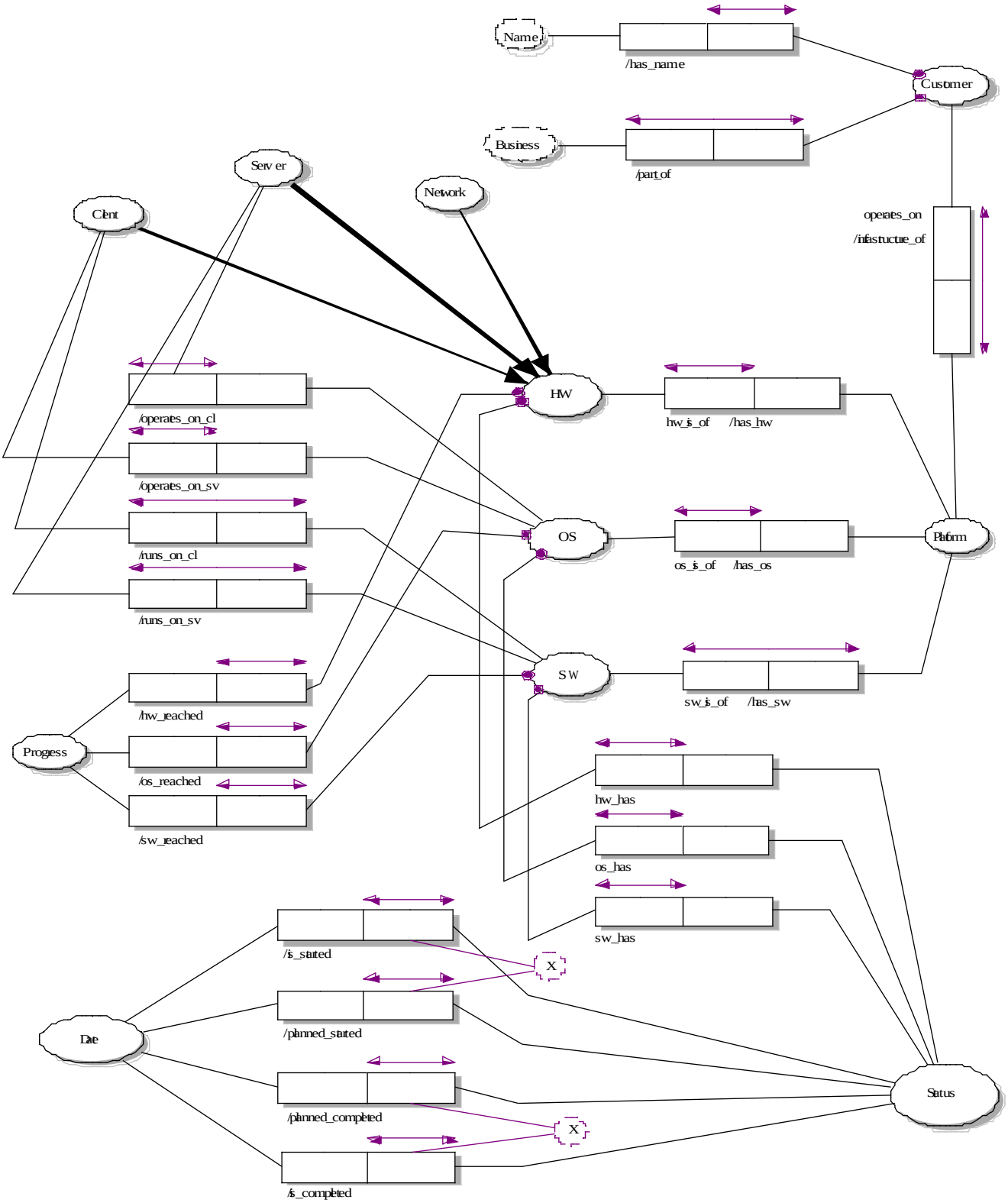

Fig. 6.1 : Content schema Y2K website.

6.2.4. The Structure of the Website

We can divide the structure of the website into three pages.

- Start page
- Customer view
- Detailed view

6.2.2.1. Start Page

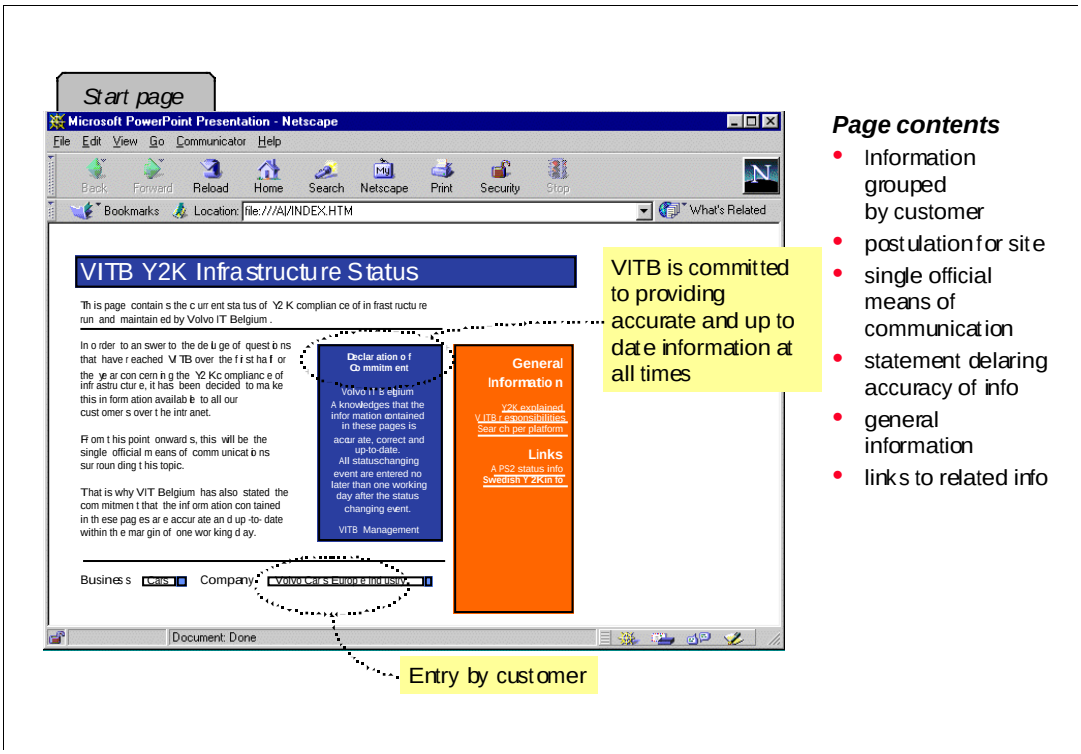

Fig. 6.2 : Start page.

This page is the central start page for every customer in search for some information concerning the Y2K status of his used infrastructure. The benefit of this centralized way of working is that each customer start from the same URL. The page contains some general and meta information like the statement concerning the commitment made by VITB regarding the statuses provided on this website. Also the links towards the Swedish Y2K site and other links are presented on this start page. Furthermore, each customer can find an entry (link) to his customer view where he can find specified information regarding his infrastructure.

6.2.2.2. Customer View

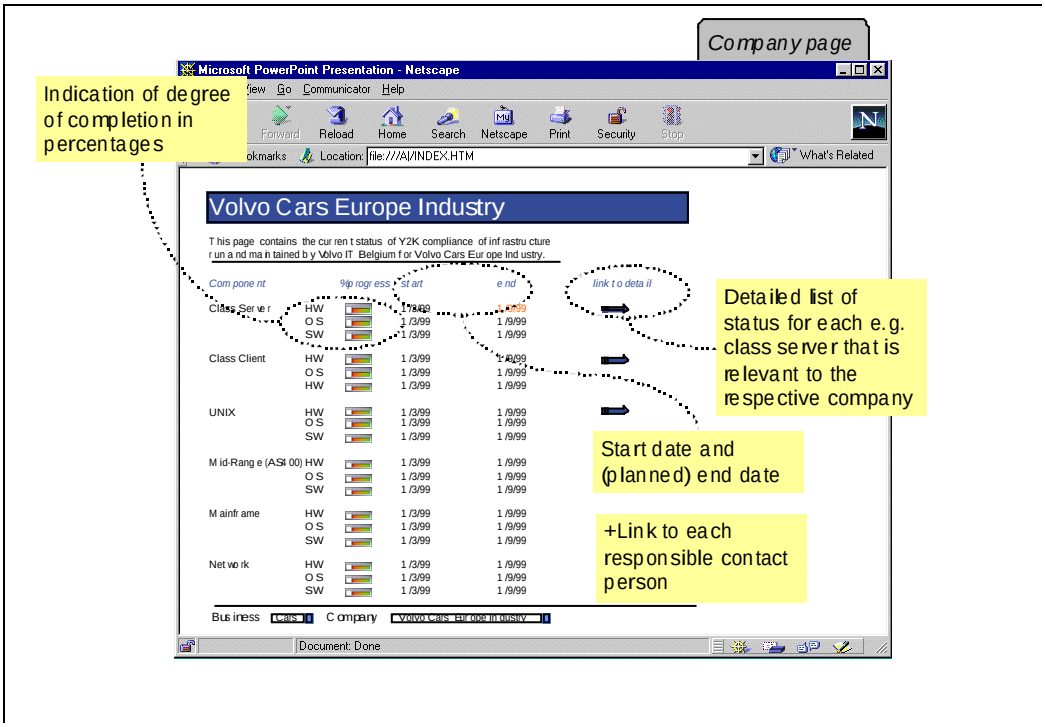

Fig. 6.3 : Customer view.

This page contains the specific information for a customer. The information is brought together in a table where it is broken down into the different platforms used by the customer. Per platform you can find an indication of the status of the Y2K compliance, divided into operating system, hardware and software. Each platform has a link towards the next page where some detailed information can be found concerning that platform.

6.2.2.3. Detailed View

There is no picture here of the detailed view. In this view, the company can see the status of all e.g. AS400 servers that are relevant to him with their respective status info. This information should be available for all “components”, Class clients, UNIX, Mid-Range, etc.

Links here are available in the table in the columns “full report”, usually leading to a PDF file and “Contact”, providing the possibility of sending e-mails to an appointed author or contact person where unanswered queries can be addressed.

6.2.5. Representation of the Structure

In this section we represent the structure of the Y2K website using the OTs and the roles declared in the meta conceptual schema and more specific the structure schema of a website represented in Fig. 4.2.
In table 6.1 we depict the OT ***Link*** and the OT ***Page*** related through the roles ***from*** and ***to***. We distinguish two Links located on the ***StartPage*** linking this page to two other pages, ***SwedishY2KPage*** and ***Apps2Page***.

Link	From Page	To Page
Swedish Y2K_link	StartPage	SwedishY2KPage
Apps2status_link	StartPage	Apps2Page

Table 6.1 : Structure elements of Y2K website.

On the other hand we have the OT ***LinkProtoType*** in relation with the OT ***Page*** or the OT ***PageProtoType***. Table 6.2 represents the two ***LinkProtoType***'s attendant in the website.

LinkProtoType	From Page	To Page	From ProtoTypePage	To ProtoTypePage
CustomerLink	StartPage			CustomerPage
PlatformLink			Customerpage	DetailedPage

Table 6.2 : Structure elements of Y2K website

As indicated in the conceptual schema in Fig 4.2 by the exclusion constraint between the roles ***from_prototype*** and ***from_p*** and the roles ***to_prototype*** and ***to_p*** a ***LinkProtoType*** can either start from a ***Page*** or a ***ProtoTypePage***, and go to a ***Page*** or a ***ProtoTypePage***. The mandatory constraint reflects that is has to be one of both each time.

We can go more in detail by representing the different instance pages of the page prototypes. E.g. we could list all the customer pages and the accompanying customer links linking them to the start page. We will leave it hereby to keep it orderly.

6.2.6. The Connection between Content and Structure

We will now use the connection schema depict in Fig. 4.3 to describe how the content of the website represented by the conceptual schema in Fig. 6.1, and more specific the OTs present in this schema, are connected with the structure elements of the website depicted in section 6.2.3.

Page	Contains Concept
StartPage	Commitment

Table 6.3 : Connection between Page and Concept.

Page	Overview of ConceptType
StartPage	Customer

Table 6.4 : Connection between Page and ConceptType.

PageProtoType	For ConceptType
CustomerPage	Customer
CustomerPage	Platform
CustomerPage	HW
CustomerPage	OS
CustomerPage	SW
CustomerPage	Status
DetailedPage	Client
DetailedPage	Server
DetailedPage	Network

Table 6.5 : Connection between PageProtoType and Concepttype

6.3. Transforming the Conceptual Schema into an XML Document

In previous section the content schema together with the structure schema and the connection schema gives a good survey how the Y2K website is built up and were we can find the different information. Also the relation between the different elements is represented by the roles they play on each other.
In this part we will transform this information into an XML document by making use of the Document type definition of section 5.1.2.

We will discuss the different elements in the document and how they are declared using the DTD. In section 6.4 you can find a more complete version of the document.

We start with the element ***Customer*** which is an instance of ***ObjectType*** and more general a ***ConceptType***. (ObjectType is declared as a sub-entity of ConceptType in the DTD).

As indicated in the algorithm described in section 5.1.1 we need to use different ID values for every element within an XML document. This is the reason why we use a different name for the ***ConceptTypeID*** and for the ***ObjectTypeID***. We adapt the ***ConceptType*** name by putting a "C" in front of the ***ObjectType*** name.

```
<ConceptType ConceptTypeID="C_Customer">
      <ObjectType ObjectTypeID="Customer">
            <with RoleTypeID="operates_on"/>
            <with RoleTypeID="has_name"/>
            <with RoleTypeID="part_of"/>
      </ObjectType>
      <with_overview_on PageID="StartPage"/>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>
```

In the content model of the element ***ObjectType*** in the DTD we find the element "***with"***, which is a reference towards a ***RoleTypeID*** and is declared to represent the roles in which the object type is playing. ***Customer*** plays three different roles in three relations with other OTs. This results in three child elements "***with*** " in the ***ObjectType*** element.

In the content model of ***ConceptType*** there are two elements, ***with_overview_on*** and ***overviewed_on*** , that indicate on which page an overview of the OT can be found and on which page prototype the details of the OT are overviewed.

As an other example we can demonstrate the OT ***Business*** for which no roles are given in the conceptual schema.

```
<ConceptType ConceptTypeID="C_Business">
      <ObjectType ObjectTypeID="Business">
      </ObjectType>
</ConceptType>
```

Next we can describe a relation declared also as a sub-element of ***ContentType*** in the DTD.

Because only the roles have a name in the conceptual schema modeled in chapter 6.2. we need to define a name convention to express the names for relations in the XML document. As an example we take the relation between Customers and Platforms consisted of the roles ***operates_on*** and ***infrastructure_of***.
We construct the name of the relation by putting together the names of the OTs playing a role in the relation. We put "C" in front of it when it is the ID value for the Concept type and "R" to indicate that it is the ID value for the RelationType.
In our example this becomes ***C_Customer_Platform*** and ***R_Customer_Platform***.

Furthermore we have one child element in the content model of ***RelationType***: ***has*** is the element which convey the reference to the roles out of which the relation is composed. We use this element to reflect the two roles ***operates_on*** and ***infrastructure_of.***

```
<ConceptType ConceptTypeID="C_Customer_Platform">
  <RelationType RelationTypeID=" R_Customer_Platform">
      <has RoleID="operates_on"/>
      <has RoleID="infrastructure_of"/>
  </RelationType>
</ConceptType>
```

When there is more then one relation between the same OTs like between ***Status*** and ***Date***, we will use the name of the most significant role together with the OT playing that role to construct the name of the relation. In this case it is obvious that we use the OT ***Status*** and the roles it play on ***Date***. E.g. ***C_Status_is_started***, ***C_Status_planned_started***, ***C_Status_planned_completed***, ***C_Status_is_completed***.

Next is a ***Role*** element including two child elements: ***played_by*** is representing the ID reference of the OT playing this role and ***in*** is the element that makes the link to the relation type element of which the role is part of.

```
<Role RoleID="operates_on">
      <played_by ObjectTypeID="Customer"/>
      <in RelationTypeID="R_Customer_Platform"/>
</Role>
```

Until now we only discussed components representing type elements. If we look at the conceptual schema in Fig 4.1 we see that this only represent the upper part of the model. At the lower side the concept element is located and is split up in the object- or relation-instances which can also be an instance of a certain type. These elements are not reflected directly in the conceptual schema of the Y2K website but are so to say at a lower level. For example as Customer objects we have Volvo trucks or Volvo Cars and many more. It is a choice how detailed you want to go in representing the website information into the XML page. At least the possibility is foreseen to do this. We will give an example of such a concept element.

```
<Concept ConceptID="C_VolvoTrucks">
      <Object ObjectID="O_VolvoTrucks">
            <ObjectInstance ObjectInstanceID="VolvoTrucks">
                  <of_objecttype ObjectTypeID="Customer"/>
            </ObjectInstance>
            <has_name> VolvoTrucks </has_name>
            <has_description> Volvo Trucks is the salescompany for
                        Volvo Trucks in Belgium.</has_description>
      </Object>
      <is_on_page PageID="P_VolvoTrucksPage"/>
</Concept>
```

Volvo Trucks is a concept, an object and an object instance of the object type ***Customer***.

The connection with the structure schema is already made by the element representing the reference towards certain pages (***is_on_page***). Next we will demonstrate how pages and links are represented in the XML document.

The ***StartPage*** is a ***Page*** and not an instance of a ***PagePrototype***.

```
<Page PageID="StartPage">
      <contains ConceptID="C_Commitment"/>
      <overview_page_of ConceptTypeID="C_Customer"/>
      <is_source_for LinkID="Swedish_Y2K_Link"/>
      <is_source_for_proto LinkProtoTypeID="CustomerLink"/>
</Page>
```

The ***CustomerPage*** can be considered as a ***PagePrototype***. As you can see, all the ***ConceptTypes*** of which there is an overview on this page are reflected by the element ***for_concept***.

```
<PagePrototype PagePrototypeID="CustomerPage">
      <for_concept ConceptTypeID="C_Customer"/>
      <for_concept ConceptTypeID="C_Platform"/>
      <for_concept ConceptTypeID="C_HW"/>
      <for_concept ConceptTypeID="C_SW"/>
      <for_concept ConceptTypeID="C_SW"/>
      <for_concept ConceptTypeID="C_Status"/>
      <for_concept ConceptTypeID="C_Progress"/>
      <is_source_proto_for LinkProtoTypeID="PlatformLink"/>
</PagePrototype>
```

We can also go in depth with the pages by defining instances of them. For example the already mentioned Volvo Trucks page being an instance of the prototype page ***CustomerPage.***

```
<Page PageID="P_VolvoTrucksPage">
      <PageInstance PageInstanceID="VolvoTrucksPage">
      <has_page_prototype PageProtoTypeID="CustomerPage"/>
      </PageInstance>
      <is_source_for LinkID="L_VolvoTruck"/>
</Page>
```

We can even go more into detail describing this page if we use the other elements out of the content model of Page.

Next we give an example of a ***LinkProtoType*** element as we can find them in the structure schema.

```
<LinkProtoType LinkProtoTypeID="CustomerLink">
      <from_p PageID="StartPage"/>
      <to_prototype PageProtoTypeID="CustomerPage"/>
</LinkProtoType>
```

Finally we present the Volvo Truck link as an instance of this link prototype between the StartPage and the ***VolvoTruck*** instance of the ***customerPage***.

```
<Link LinkID="L_VolvoTruck">
      <LinkInstance LinkInstanceID="VolvoTruck">
      <has_link_prototype LinkProtoTypeID="CustomerLink"/>
      </LinkInstance>
      <from PageID="StartPage"/>
      <to PageID="P_VolvoTrucksPage"/>
</Link>
```

6.4. XML Document of the Volvo IT Belgium Y2K Website

In this section we present the complete XML document of the Volvo IT Y2K website. You will notice that we can go much more in detail representing the information in the website. We only gave a example with Volvo Truck as an instance of a customer. Nevertheless the possibility exist to record every instance of whatever concept exists in the website. To preserve an overview we did not go deeper into it.

```
<?xml version="1.0"?>
<!DOCTYPE Website SYSTEM "website.dtd">

<Website WebsiteID="Y2Kwebsite">


<!--                    ConceptType                              -->

<ConceptType ConceptTypeID="C_Customer">
      <ObjectType ObjectTypeID="Customer">
            <with RoleTypeID="operates_on"/>
            <with RoleTypeID="has_name"/>
            <with RoleTypeID="part_of"/>
      </ObjectType>
      <with_overview_on PageID="StartPage"/>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Name">
      <ObjectType ObjectTypeID="Name">
      </ObjectType>
</ConceptType>

<ConceptType ConceptTypeID="C_Business">
      <ObjectType ObjectTypeID="Business">
      </ObjectType>
      <with_overview_on PageID="StartPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Platform">
      <ObjectType ObjectTypeID="Platform">
            <with RoleTypeID="infrastructure_of"/>
            <with RoleTypeID="has_hw"/>
            <with RoleTypeID="has_os"/>
            <with RoleTypeID="has_sw"/>
      </ObjectType>
      <with_overview_on PageID="StartPage"/>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_HW">
      <ObjectType ObjectTypeID="HW">
            <with RoleTypeID="hw_reached"/>
            <with RoleTypeID="hw_has"/>
            <with RoleTypeID="hw_is_of"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>
```

```
<ConceptType ConceptTypeID="C_SW">
      <ObjectType ObjectTypeID="SW">
            <with RoleTypeID="sw_reached"/>
            <with RoleTypeID="sw_has"/>
            <with RoleTypeID="sw_is_of"/>
            <with RoleTypeID="runs_on_cl"/>
            <with RoleTypeID="runs_on_sv"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_OS">
      <ObjectType ObjectTypeID="OS">
            <with RoleTypeID="os_reached"/>
            <with RoleTypeID="os_has"/>
            <with RoleTypeID="os_is_of"/>
            <with RoleTypeID="operates_on_cl"/>
            <with RoleTypeID="operates_on_sv"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Network">
      <ObjectType ObjectTypeID="Network">
            <is_sub ObjectTypeID="HW"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="DetailedPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Client">
      <ObjectType ObjectTypeID="Client">
            <is_sub ObjectTypeID="HW"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="DetailedPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Server">
      <ObjectType ObjectTypeID="Server">
            <is_sub ObjectTypeID="HW"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="DetailedPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Status">
      <ObjectType ObjectTypeID="Status">
            <with RoleTypeID="is_started"/>
            <with RoleTypeID="planned_started"/>
            <with RoleTypeID="planned_completed"/>
            <with RoleTypeID="is_completed"/>
      </ObjectType>
      <overviewed_on PageProtoTypeID="CustomerPage"/>
</ConceptType>

<ConceptType ConceptTypeID="C_Date">
      <ObjectType ObjectTypeID="Date">
      </ObjectType>
</ConceptType>

<ConceptType ConceptTypeID="C_Progress">
      <ObjectType ObjectTypeID="Progress">
```

```
      </ObjectType>
</ConceptType>


<!--                RelationTypes                          -->

<ConceptType ConceptTypeID="C_Customer_Platform">
  <RelationType RelationTypeID=" R_Customer_Platform">
      <has RoleID="operates_on"/>
      <has RoleID="infrastructure_of"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Customer_Name">
  <RelationType RelationTypeID=" R_Customer_Name">
      <has RoleID="has_name"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Customer_Business">
  <RelationType RelationTypeID=" R_Customer_Business">
      <has RoleID="part_of"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Platform_HW">
  <RelationType RelationTypeID=" R_Platform_HW">
      <has RoleID="has_hw"/>
      <has RoleID="hw_is_of"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Platform_OS">
  <RelationType RelationTypeID=" R_Platform_OS">
      <has RoleID="has_os"/>
      <has RoleID="os_is_of"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Platform_SW">
  <RelationType RelationTypeID=" R_Platform_SW">
      <has RoleID="has_sw"/>
      <has RoleID="sw_is_of"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_SW_Client">
  <RelationType RelationTypeID=" R_SW_Client">
      <has RoleID="runs_on_cl"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_SW_Server">
  <RelationType RelationTypeID=" R_SW_Server">
      <has RoleID="runs_on_sv"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_OS_Client">
  <RelationType RelationTypeID=" R_OS_Client">
      <has RoleID="operates_on_cl"/>
```

```
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_OS_Server">
  <RelationType RelationTypeID=" R_OS_Server">
      <has RoleID="operates_on_sv"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Status_is_started">
  <RelationType RelationTypeID=" R_Status_is_started">
      <has RoleID="is_started"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Status_is_completed">
  <RelationType RelationTypeID=" R_Status_is_completed">
      <has RoleID="is_completed"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Status_planned_started">
  <RelationType RelationTypeID=" R_Status_planned_started">
      <has RoleID="planned_started"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_Status_planned_completed">
  <RelationType RelationTypeID=" R_Status_planned_completed">
      <has RoleID="planned_completed"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_SW_Progress">
  <RelationType RelationTypeID=" R_SW_Progress">
      <has RoleID="sw_reached"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_SW_Status">
  <RelationType RelationTypeID=" R_SW_Status">
      <has RoleID="sw_has"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_OS_Progress">
  <RelationType RelationTypeID=" R_OS_Progress">
      <has RoleID="os_reached"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_OS_Status">
  <RelationType RelationTypeID=" R_OS_Status">
      <has RoleID="os_has"/>
  </RelationType>
</ConceptType>

<ConceptType ConceptTypeID="C_HW_Progress">
  <RelationType RelationTypeID=" R_HW_Progress">
      <has RoleID="hw_reached"/>
  </RelationType>
```

```
</ConceptType>

<ConceptType ConceptTypeID="C_HW_Status">
  <RelationType RelationTypeID=" R_HW_Status">
      <has RoleID="hw_has"/>
  </RelationType>
</ConceptType>


<!--                    Roles                              -->

<Role RoleID="operates_on">
      <played_by ObjectTypeID="Customer"/>
      <in RelationTypeID="R_Customer_Platform"/>
</Role>

<Role RoleID="has_name">
      <played_by ObjectTypeID="Customer"/>
      <in RelationTypeID="R_Customer_Name"/>
</Role>

<Role RoleID="part_of">
      <played_by ObjectTypeID="Customer"/>
      <in RelationTypeID="R_Customer_Business"/>
</Role>

<Role RoleID="infrastructure_of">
      <played_by ObjectTypeID="Platform"/>
      <in RelationTypeID="R_Customer_Platform"/>
</Role>

<Role RoleID="has_hw">
      <played_by ObjectTypeID="Platform"/>
      <in RelationTypeID="R_Platform_HW"/>
</Role>

<Role RoleID="has_os">
      <played_by ObjectTypeID="Platform"/>
      <in RelationTypeID="R_Platform_OS"/>
</Role>

<Role RoleID="has_sw">
      <played_by ObjectTypeID="Platform"/>
      <in RelationTypeID="R_Platform_SW"/>
</Role>

<Role RoleID="hw_is_of">
      <played_by ObjectTypeID="HW"/>
      <in RelationTypeID="R_Platform_HW"/>
</Role>

<Role RoleID="os_is_of">
      <played_by ObjectTypeID="OS"/>
      <in RelationTypeID="R_Platform_OS"/>
</Role>

<Role RoleID="sw_is_of">
      <played_by ObjectTypeID="SW"/>
      <in RelationTypeID="R_Platform_SW"/>
</Role>
```

```
<Role RoleID="runs_on_cl">
      <played_by ObjectTypeID="SW"/>
      <in RelationTypeID="R_SW_Client"/>
</Role>

<Role RoleID="runs_on_sv">
      <played_by ObjectTypeID="SW"/>
      <in RelationTypeID="R_SW_Server"/>
</Role>

<Role RoleID="operates_on_sv">
      <played_by ObjectTypeID="OS"/>
      <in RelationTypeID="R_OS_Server"/>
</Role>

<Role RoleID="operates_on_cl">
      <played_by ObjectTypeID="OS"/>
      <in RelationTypeID="R_OS_Client"/>
</Role>

<Role RoleID="is_started">
      <played_by ObjectTypeID="Status"/>
      <in RelationTypeID="R_Status_is_started"/>
</Role>

<Role RoleID="is_completed">
      <played_by ObjectTypeID="Status"/>
      <in RelationTypeID="R_Status_is_completed"/>
</Role>

<Role RoleID="planned_started">
      <played_by ObjectTypeID="Status"/>
      <in RelationTypeID="R_Status_planned_started"/>
</Role>

<Role RoleID="planned_completed">
      <played_by ObjectTypeID="Status"/>
      <in RelationTypeID="R_Status_planned_completed"/>
</Role>

<Role RoleID="sw_reached">
      <played_by ObjectTypeID="SW"/>
      <in RelationTypeID="R_SW_Progress"/>
</Role>

<Role RoleID="sw_has">
      <played_by ObjectTypeID="SW"/>
      <in RelationTypeID="R_SW_Status"/>
</Role>

<Role RoleID="os_reached">
      <played_by ObjectTypeID="OS"/>
      <in RelationTypeID="R_OS_Progress"/>
</Role>

<Role RoleID="os_has">
      <played_by ObjectTypeID="OS"/>
      <in RelationTypeID="R_OS_Status"/>
</Role>

<Role RoleID="hw_reached">
```

```
      <played_by ObjectTypeID="HW"/>
      <in RelationTypeID="R_HW_Progress"/>
</Role>

<Role RoleID="hw_has">
      <played_by ObjectTypeID="HW"/>
      <in RelationTypeID="R_HW_Status"/>
</Role>


<!--                    Pages                            -->

<Page PageID="StartPage">
      <contains ConceptID="C_Commitment"/>
      <overview_page_of ConceptTypeID="C_Customer"/>
      <is_source_for LinkID="Swedish_Y2K_Link"/>
      <is_source_for_proto LinkProtoTypeID="CustomerLink"/>
</Page>

<Page PageID="SwedishY2KPage">
      <is_target_for LinkID="Swedish_Y2K_Link"/>
</Page>

<Page PageID="P_VolvoTrucksPage">
      <PageInstance PageInstanceID="VolvoTrucksPage">
      <has_page_prototype PageProtoTypeID="CustomerPage"/>
      </PageInstance>
      <is_source_for LinkID="L_VolvoTruck"/>
</Page>


<!--                    PagesProtoType                     -->

<PageProtoType PageProtoTypeID="CustomerPage">
      <for_concept ConceptTypeID="C_Customer"/>
      <for_concept ConceptTypeID="C_Platform"/>
      <for_concept ConceptTypeID="C_HW"/>
      <for_concept ConceptTypeID="C_SW"/>
      <for_concept ConceptTypeID="C_SW"/>
      <for_concept ConceptTypeID="C_Status"/>
      <for_concept ConceptTypeID="C_Progress"/>
      <is_source_proto_for LinkProtoTypeID="PlatformLink"/>
</PageProtoType>

<PageProtoType PageProtoTypeID="DetailedPage">
      <for_concept ConceptTypeID="C_Network"/>
      <for_concept ConceptTypeID="C_Client"/>
      <for_concept ConceptTypeID="C_Server"/>
      <is_target_proto_for LinkProtoTypeID="PlatformLink"/>
</PageProtoType>


<!--                    LinkProtoTypeID                          -->

<LinkProtoType LinkProtoTypeID="CustomerLink">
      <from_p PageID="StartPage"/>
      <to_prototype PageProtoTypeID="CustomerPage"/>
</LinkProtoType>

<LinkProtoType LinkProtoTypeID="PlatformLink">
      <from_prototype PageProtoTypeID="CustomerPage"/>
```

```
      <to_prototype PageProtoTypeID="DetailedPage"/>
</LinkProtoType>



<!--                    Link                              -->

<Link LinkID="Swedish_Y2K_Link">
      <from PageID="StartPage"/>
      <to PageID="SwedishY2KPage"/>
</Link>

<Link LinkID="L_VolvoTruck">
      <LinkInstance LinkInstanceID="VolvoTruck">
      <has_link_prototype LinkProtoTypeID="CustomerLink"/>
      </LinkInstance>
      <from PageID="StartPage"/>
      <to PageID="P_VolvoTrucksPage"/>
</Link>


<!--                    Concept                             -->

<Concept ConceptID="C_Commitment">
      <Object ObjectID="Commitment">
      </Object>
</Concept>

<Concept ConceptID="C_VolvoTrucks">
      <Object ObjectID="O_VolvoTrucks">
            <ObjectInstance ObjectInstanceID="VolvoTrucks">
                  <of_objecttype ObjectTypeID="Customer"/>
            </ObjectInstance>
            <has_name> VolvoTrucks </has_name>
            <has_description> Volvo Trucks is the salescompany for
Volvo Trucks in Belgium.</has_description>
      </Object>
      <is_on_page PageID="P_VolvoTrucksPage"/>
</Concept>


</Website>
```

7. Utilization of the XML Documents of Websites

In this chapter we will illustrate what can be done with the XML documents of a website constructed in accordance with the DTD of chapter 5. We will show how the information can be extracted from the XML document using the XML Document Object Model (DOM) [BD 98]. In a small example this information will be reflected and made useful to the user of the website of which the XML document represents the conceptual schema. This can be used as a guide map to find and work with the information available in the website. Most of the professional websites nowadays use some sort of a sitemap which is no more then a list of all the subjects present in the website, and a link from every subject to a certain page. We will demonstrate how the XML document can be used to create some kind of a sitemap, but with a lot more information to provide. The lost in hyperspace syndrome can be avoided in this way. It should be clear that what we give here is only an illustration of the possibilities. More sophisticated representations and application are possible, e.g. a graphical representation of the conceptual schema of the website.

7.1. The XML Document Object Model

The XML Document Object Model (DOM) proposed by the World Wide Web Consortium (W3C) is potentially one of the most important standards since the XML specification. It gives implementers a common vocabulary to use in manipulating the XML document. No matter which language is applied, exactly the same methods can be used. To accomplish a task, exactly the same commands can be given whether the application was written in C, Java, Python or Visual Basic. We will first look at the concept of the DOM as applied to XML documents. The core model proposed by the W3C consists of a suggested API (Application Program Interface) for various different applications. The importance of this common API originates in the fact that it allows programmers to use the same commands to accomplish the same task in any collaborating application. This of course provided the different vendors agree to stick to it. It almost looks as if with XML this can be reached because all the major vendors have promised to adhere to the W3C DOM interface. Internet Explorer 5 has already implemented the majority of it. We will use it as an application to test and run the examples we will develop to explore the XML documents.

7.1.1. An XML Document as an Object Collection

When we consider an XML document we can immediately define some objects and their properties. To illustrate some of these objects we can use the following example:

```
<Website WebsiteID="Y2Kwebsite">
      <ConceptType ConceptTypeID="C_Customer">
            <ObjectType ObjectTypeID="Customer">
                  <with RoleTypeID="operates_on"/>
```

```
                    <with RoleTypeID="has_name"/>
                    <with RoleTypeID="part_of"/>
                </ObjectType>
                <with_overview_on PageID="StartPage"/>
                <overviewed_on PageProtoTypeID="CustomerPage"/>
        </ConceptType>
</Website>
```

We distinguish in the first place the ***Document*** object itself which includes the whole document. The ***Website*** object is an element object that has one child ***ConceptTyp***e.
In the element object ***ConceptType*** there are three children : ***ObjectType***, ***with_overview_on*** and ***overviewed_on***. These objects are sibling element objects.
In XML there are some other types of objects such as comments, XML declarations and DTDs.

Each of these objects can be described and portrayed by the use of some properties. These properties should make it possible to build up a document only by using the description of the properties of the different objects. This way a document can be rebuilt over and over again when the information about the properties is available for each object.

Extending our analogy to the nodes of XML documents, here are some of the properties that are included:

- **Type** : This property gives information regarding the type of node, like comment, element, text node, etc. We will see later that types are presented by a number (1 = element node) and that they can be called for using a certain method of the node interface.
- **Value** : A value is related to the nodes content.
- **Name :** The name of an element node is its tag name.
- **Attributes** : A list of attribute/value pairs of the element. Attributes could be handled as child nodes, however the W3C chooses to treat them as properties of the element node.
- **Parent** : The parent of the current node.
- **Siblings** : A knowledge of its position in a list of siblings, or a knowledge of who its elder and younger siblings are.

The DOM presents documents as a hierarchy of node objects that implement interfaces. This is completely in line with the Object Oriented way of working [TB 97]. In OO languages like C++ we also see the use of objects and the methods that can be executed on them. The group of methods is called the interface of the object.

7.1.2. The DOM Interfaces

In this section we will discuss some interfaces and their methods to illustrate how these tools can be used to manipulate a document.

We will start by looking at the Node interface. All the node types in the DOM inherit a basic set of properties and methods from the node interface. We can make a difference between methods providing attributes and factory methods.

Attribute methods:

These are methods like ***nodeName*** or ***nodeValue*** giving back the name or the value of the node on which it is used. ***NodeName*** results into a string presenting the name of the node. When a node is an element it corresponds with the tag name, when it is a text node it will have the value #text.
Depending on the method the result can either be a string, a node, a nodelist, or even a document or an unsigned short representing a type of node.
The use of these methods is of the same nature as the use of methods in other OO program languages. The object is used with the method placed after it with a point to connect them.

```
var x=myDoc.childNodes;
            var z=x.length;
  for(var i=0; i < z ; i++)
      {
      document.write(i + " ___  " + x(i).nodeType+ "___ ");
      document.write( x(i).nodeName+ "___ ");
      document.write( x(i).nodeValue + "<BR> ");
      }
```

Note that in the example, ***myDoc*** is the document object which we can manipulate using the DOM. This example is depicted completely in Appendix B, example 1. There you can see how the document object is created by loading the XML document into this document object. We make use of the combination of HTML and JavaScript to work with the DOM and to make it presentable for browsers. This way it can be sent around the Internet. In Fig 7.1 the result of the HTML page used with the XML document of the Y2K website of chapter 6 is reflected.

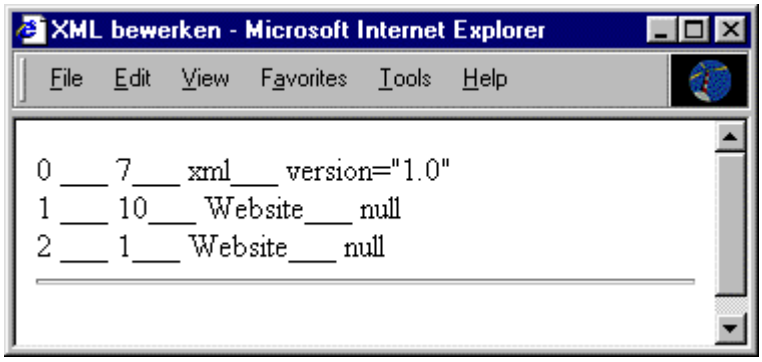

Fig. 7.1 : Result of using attribute methods in the XML DOM.

In the code the attribute method ***childNodes*** is used on the node ***mydoc***. The ***childNodes*** attribute returns a ***Nodelist*** object of all the children of any given node. This is a list holding all the properties of the children of the node on which it is executed. We can explore the list through iteration. The different children can be accessed using an index.
In the iteration we demonstrate the different attribute methods that can be used on the different children. In Fig. 7.1 we see the result of this iteration executed on the XML document of the Y2K website (section 6.4). Only three nodes are reflected in the result. These are the three children nodes of the document. In the second column we see the number corresponding with the type of the node. Number 7 indicates a processing instruction node which correspond with the node <?xml version="1.0"?>, number 10 indicates a document type node corresponding wit the node <!DOCTYPE …>, while number 1 indicates an element node. The element node is the <website …> node which is the top element of the document. The third and fourth column are the resul of the methods ***nodeName*** and ***nodeValue.***

Factory methods:

The node factory methods allow us to insert the nodes we have already created with the document. In contrast to the attribute methods the factory methods allow us to make changes to the document. E.g. ***insertBefore*** takes two parameters, the node object to insert and the node object to insert it before. It returns the node being inserted.
The document interface factory methods make it possible to create each and every kind of node. E.g. ***createElement*** is the method to create an element, it takes a string as parameter which represent the tag name of the element.

We find al sorts of methods in the different interfaces of the different objects that are defined in the XML DOM. More information can be found in specialized literature [BD 98] and on the Microsoft XML website.

7.2. Exploring the Conceptual Schema of a Websites with the XML DOM

We will use the explained theory about the XML DOM together with JavaScript [NN 96] to build some rather simple programs and include them into HTML pages. This way they can be easily transported over the Internet. The examples will illustrate how these techniques can be used to manipulate XML pages.

7.2.1. Counting the different ObjectTypes

In this example we use the XML DOM to explore an XML document in a very superficial way. As a result of the function count() the number of different ObjectTypes is displayed. Figure 7.2 depicts the result when the program is used on the XML document of the Y2K website (chapter 6).

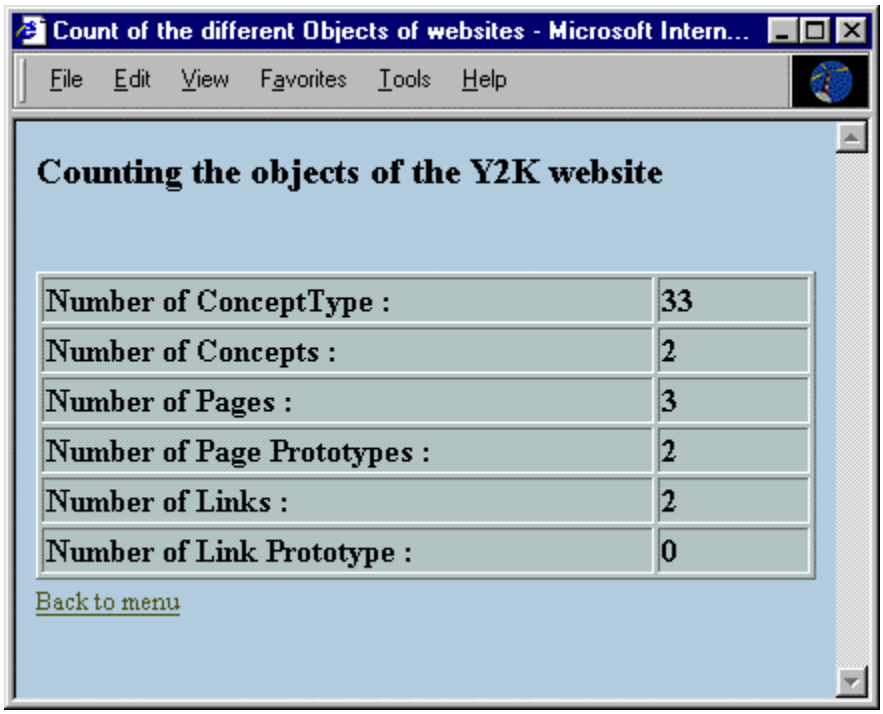

Fig. 7.2 : Result of counting the Objects of the Y2K website.

The complete listing of the example can be found in Appendix B, example 2. The main use of the XML DOM is in the statement ***getElementByTagName*** executed on the document node to search for specific elements.

```
var concepttypes = myDoc.getElementsByTagName("ConceptType");
```

The result is put into a variable and is a list of all the elements of which the name is similar as the parameter in the method. The length of this list reflects the number of elements it has found.

7.2.2. Reflecting the Elements of the Conceptual Schema

The example 3 until 5 are programs that return information regarding a specific element of the conceptual schema of websites transformed into the XML document.

The ***ConceptType*** in the XML document can be explored by using example 3. In Fig. 7.3 the result of the first concept type is presented. By scrolling down, all the Concept Types can be explored. In this image you can find the different elements out of which a ***ConceptType*** is built up and the information they reveal. The first ***ConceptType*** is a ***Customer*** playing three roles in the conceptual schema. It also reveals on what

prototype page a customers information is represented and on what page an overview of all customers can be found.

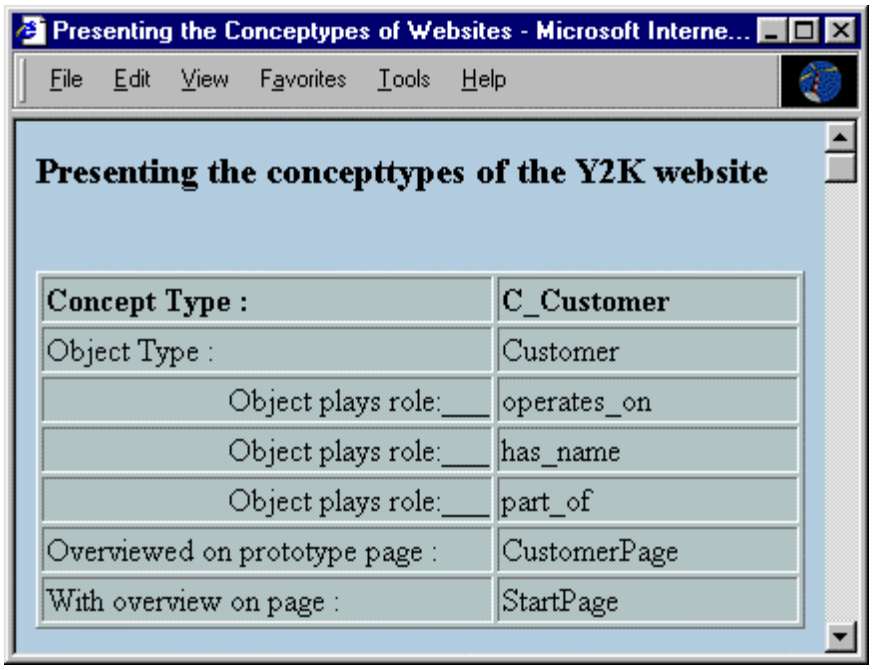

Fig. 7.3 : Result of exploring the ConceptTypes of the Y2K website.

By using the other examples you can find the details concerning the Pages and the Links, also reflecting the different elements and the information they contain. Fig. 6.3 presents the result of example 4 and the first page.

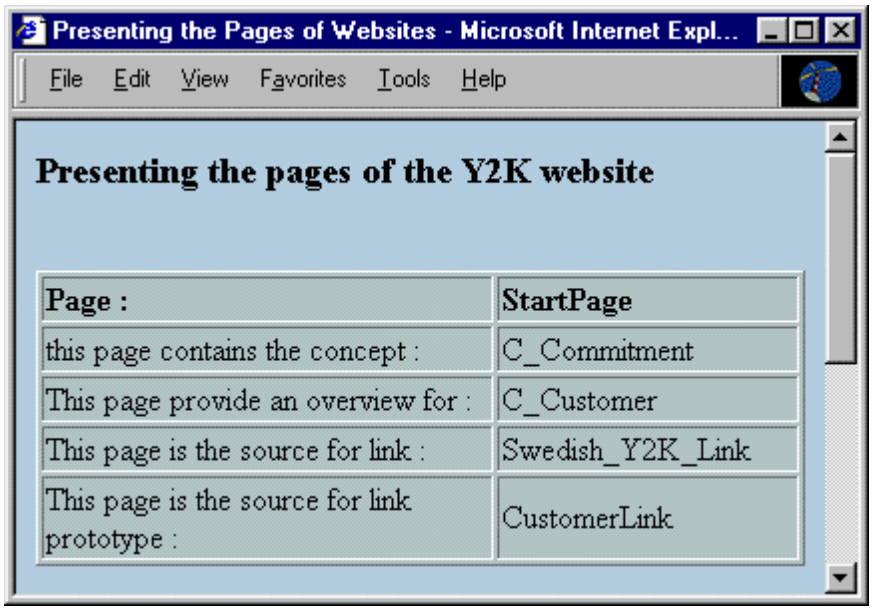

Fig. 7.4 : Result of exploring the Pages of the Y2K website.

7.2.3. Revealing the Relations of ObjectTypes within the Conceptual Schema.

In this last example 6 we use different elements and the references created by the different IDattributes to rebuild and represent the relations of the different Object Types. When the HTML page is selected, the user will be asked to put in an ***ObjectType*** of which he wants to explore the roles it plays in the different relations.

As an example we use the program with the OT ***Customer***. Figure 7.4 depicts a part of the content conceptual schema of the Y2K website (Fig. 6.1) in which the Customer is present, whereas Fig. 7.5 reflects how this is transformed using the code of example 6.

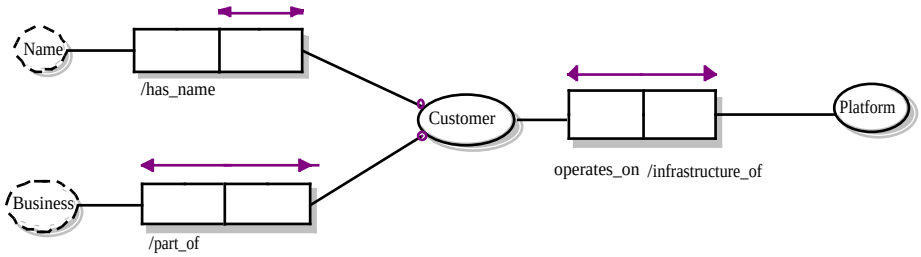

Fig. 7.5 : Part of the conceptual schema of the Y2K website containing the Customer relations.

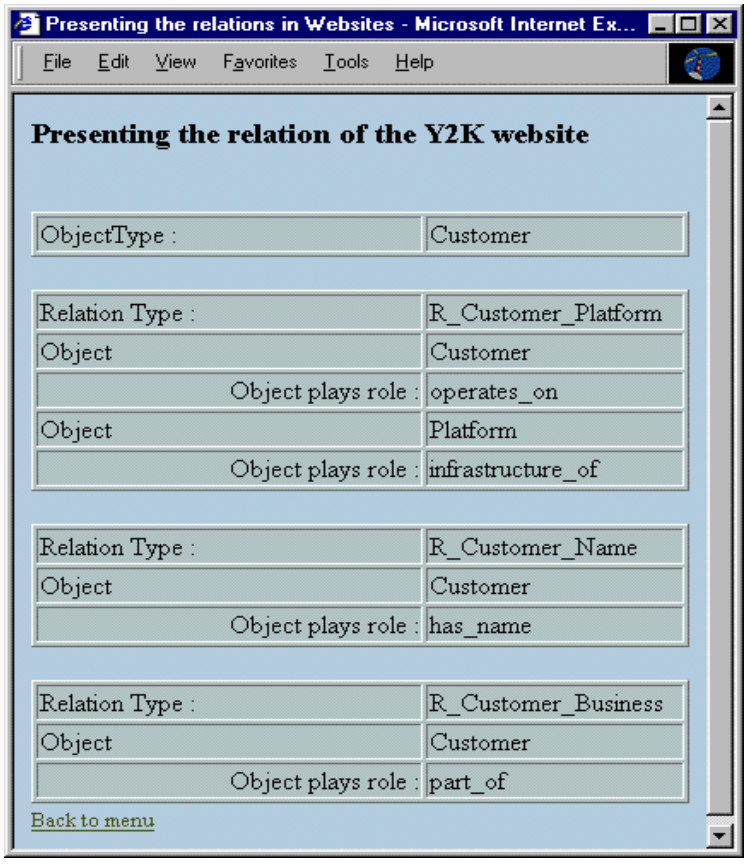

Fig. 7.6 : Presentation of the Customer relations in the Y2K website.

This way of presenting the information is a good method of approach for the graphical representation as in E.g. Fig. 6.1.

7.2. Next Step

In a next step it should be possible to extend the way we transform the conceptual schema into a DTD. More information could be included in the different elements. In this thesis we only worked with elements where most of the information is stored in the attributes of the different elements. We did not include any data between the tags of an element of the XML document.
One of the possibilities could be that together with every instance of a certain OT a URL to a web page is given on which this instance is actually mentioned. If we present the conceptual schema of the website as illustrated in this chapter, the XML DOM can then be used to manipulate this reference and reflect it together with the object it belongs to. This can be done with the help of the reference tag <a> in HTML so that the instance of the object type is the representation of the link towards that page. Also listings of all available instances could be included into an element as normal text. This can be used to display extra information of an OT.
As already mentioned the given examples can be extended in a graphical way presenting the conceptual schema of a website like displayed in the different figures made with ObjectModeler (E.g. Fig. 6.1). We could extend this graphical representation by including references to the different OTs present in the schema. This way users of the schema can click on an element and immediately go to the page where it is attendant or extra information like a list of all the instances of the OT can be revealed.
Concerning the maintenance of the XML document and adapting it according to changes to the website it stands for, we can make use of the factory methods as discussed in section 7.1.2. After all the factory methods are able to make changes to the XML documents and create new elements into it. It should be possible to create a interface program that gives an administrator of a website the possibility to adapt the XML document in a user friendly way without having to work in the XML document itself.

Conclusion

Looking at the evolution of the Internet it will become more and more important to introduce some structure and standardization. This to optimize the use of this pile of data and make it more professional to exploit. We examined in this thesis the possibilities XML holds to realize some of these objectives.
As a general conclusion of this thesis we can say that in our search for a common way to bring more structured information concerning the structure and content of a website, we proved that XML is a good tool to present conceptual schemas of websites. Furthermore XML has the capacity to work with and manipulate the information enclosed in the XML documents using the standard XML DOM.
We succeeded in creating an algorithm to transform every conceptual schema into a DTD and used it to create a DTD of the conceptual schema of a website. This gave us a standard frame to use with the XML DOM and made it possible to manipulate and use the XML documents built up according this DTD. The communality between these documents can be used to develop programs to manipulate the included information. If we use this theory it should be possible to have for every website an XML document containing the necessary information to reproduce the conceptual schema of the particular website. The use of the XML DOM in combination with Java Script makes it possible to present the information available in the XML document as illustrated in Chapter 7. This information could be made accessible through a link from the homepage of every website. It will provide the users and visitors of the website an instrument to explore the data available inside the website.
We can conclude by saying that we demonstrated that XML is an opportunity to help the exploitation of the Internet to become more professional.

Appendix A

Volvo IT Belgium, Y2K Status Information, situation sketch.

What is VITB

Volvo Information technology Belgium (VITB) is a regional center which is part of the global organization Volvo Information Technology. The business activities are mainly operations and infrastructures for companies within the Volvo organization located in Belgium, the Netherlands, France, Italy and Spain. These companies which are the customers of VITB are located in all sorts of business areas, for example production plants as Volvo Cars Europe Industry in Gent, Marketing departments, sales companies, warehouses, and action centers. These customers use the provided infrastructure going from Mainframe through Mid range (AS400) and client server applications.
Today some 100 people are working in this center and this amount is increasing every quarter. The scope of the business is also growing towards application development and support.
One of the main project at this moment is the Y2K compliance of all hardware and software within the responsibility of VITB. The case treated in this sections deals with the communication aspect within this project.

Problem description

In order to answer to the deluge of queries from customers concerning the Y2K compliance of servers run and maintained by Volvo IT Belgium, the need for a centralized means of communication was identified. There is need for a forum that unambiguously and easily responds to the information needs on the customer side.
The target groups are all Volvo customers of Volvo IT Belgium that rely on the company to assure that the infrastructure they use, for which VITB bears the responsibility, is Y2K compliant by the year-end.

Choice of medium

The choice of medium must be founded on a number of criteria that it must comply to and have been identified as vital in this area:

- Accessible to all customers
- Accurate and punctual
- Low cost, easy to update
- Part of existing communication lines
- Accepted as official means of communication
- Binding nature and perceived credibility

The medium best answering to the criteria stipulated above is an electronic one. By using the existing electronic communication lines inside Volvo, the recommendation goes out to the creation of a centrally controlled website, formally stating the desired information in a clear and easy way.
But even though this medium may hold the most intrinsically interesting characteristics, it is still no guarantee for success. It must be utilized in the best possible way and answer to some strict conditions as specified below:

- **Accessibility for all customers**
  The site must be situated in an environment that can be reached by all customers. But the successful offering of information is not just the presence of availability, it is the knowledge of existence. It is of the utmost in importance that the target audience is notified of the existence of the new site and that the information there is considered to be the only reference point. If the customer is not aware that the information is there, the medium does not comply with the criteria of accessibility, as it remains cognitively hidden.
- **Accurate en punctual**
  The one appointed for the maintenance of the site must take responsibility and assure that the site always contains the latest information. Not only is it important to have a site that is up-to-date at all times, responsibility and accountability must also be specified. Upon publishing, the one responsible should be fully aware of the information's accuracy and its binding nature.
  The advice is to ensure a time lapse of no more than one working day after a status-changing event took place. The page should therefore always state the date at which it was last updated.
- **Low cost**

The cost for spreading information and making it available over the intranet is intrinsically low. To also control and minimize the number of man-hours spent on maintaining the site, a routine should be created that is as simple as possible but as controlled as necessary, considering the binding nature of the information. The flow of information should be as flat as possible and the number of intermediate steps as few as the organization permits.

- **Make part of the existing communication lines**
  To save time and effort the message must be conveyed over the existing lines of communication. The intranet is then the obvious choice in view of its omnipresence among the customers.
- **Accepted as official means of communications**
  The information must be presented as originating from the official organ that has specified the commitment to ensure the Y2K compliance. In that sense, the page must be incorporated into the existing VITB web environment.
  The risk here however is, that if the rest of the environment does not possess the characteristics of being accurate and punctual, this could negatively affect the ambition of the information. In other words, if the environment does not look or feel official enough and is not being maintained properly, the credibility of this page could well be questioned from the part of the receiver.
- **Binding nature and perceived credibility**
  This is foremost the weakest characteristic of an on-line environment. To what extent can the information on a web page be declared to offer the same kind of commitment as a statement on paper? Because this question will equally play in the heads of the customers and because many of them will have experiences with appalling out-of-date websites, it is of the utmost importance to clearly address this subject and issue a statement about it. This should be a kind of declaration of commitment stating that VITB acknowledges that the information contained in the page is accurate at the publication date, and that the company commits to having offered correct information.

Appendix B

Example 1

```
<HTML>
<HEAD>
<TITLE> Manipulating XML </TITLE>
</HEAD>
<XML ID="xmldata" ></XML>

<SCRIPT>
  xmldata.load("website.xml");
  var myDoc=xmldata;
  if (myDoc.parseError.reason !="")
       {
       alert(myDoc.parseError.reason)
       }

  var x=myDoc.childNodes;
               var z=x.length;
  for(var i=0; i < z ; i++)
       {
       document.write(i + " ___  " + x(i).nodeType+ "___ ");
       document.write( x(i).nodeName+ "___ ");
       document.write( x(i).nodeValue + "<BR> ");
       }

document.write("<hr size=3>" + "<BR> ");

</SCRIPT>

</HTML>
```

Example 2

```
<HTML>
<HEAD>
<TITLE> Count of the different Objects of websites</TITLE>
</HEAD>
<body bgcolor="#BCD3E">
<XML ID="xmldoc" ></XML>
<h3> Counting the objects of the Y2K website</h3>
<SCRIPT>

function load() {
  xmldoc.load("website.xml");

  }
function contentcount() {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
        {
        alert(myDoc.parseError.reason)
        }

  var concepttypes = myDoc.getElementsByTagName("ConceptType");
  var concept = myDoc.getElementsByTagName("Concept");
  var page = myDoc.getElementsByTagName("Page");
  var role = myDoc.getElementsByTagName("Role");
  var pageprototype = myDoc.getElementsByTagName("PageProtoType");
  var link = myDoc.getElementsByTagName("Link");
  var linkprototype = myDoc.getElementsByTagName("Linkprototype");

  var number_concepttypes = concepttypes.length;
  var number_roles = role.length;
  var number_pages = page.length;
  var number_concept = concept.length;
  var number_pageprototypes = pageprototype.length;
  var number_links = link.length;
  var number_linkprototypes = linkprototype.length;

  document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

  document.write("<tr><td width=\"80\%\"><b>Number of ConceptType : </b></td><td
width=\"20\%\"><b> " + number_concepttypes + "</b></td></tr>");

  document.write("<tr><td width=\"80\%\"><b>Number of Concepts : </b></td><td
width=\"20\%\"><b> " + number_concept + "</b></td></tr>");

  document.write("<tr><td width=\"80\%\"><b>Number of Pages : </b></td><td
width=\"20\%\"><b> " + number_pages + "</b></td></tr>");

  document.write("<tr><td width=\"80\%\"><b>Number of Page Prototypes : </b></td><td
width=\"20\%\"><b> " + number_pageprototypes + "</b></td></tr>");

  document.write("<tr><td width=\"80\%\"><b>Number of Links : </b></td><td
width=\"20\%\"><b> " + number_links + "</b></td></tr>");

  document.write("<tr><td width=\"80\%\"><b>Number of Link Prototype : </b></td><td
width=\"20\%\"><b> " + number_linkprototypes + "</b></td></tr>");

  document.write("</table>");

}
contentcount()

</SCRIPT>

<a href="schema1.htm" ><font
        size="2">Back to menu</font></a></

</body>
</HTML>
```

Example 3

```
<HTML>
<HEAD>
<TITLE> Presenting the Conceptypes of Websites </TITLE>
</HEAD>
<body bgcolor="#BCD3E">
<XML ID="xmldoc" ></XML>
<h3> Presenting the concepttypes of the Y2K website</h3>
<SCRIPT>
function load() {
  xmldoc.load("website.xml");

  }

function concepttypes() {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
        {
        alert(myDoc.parseError.reason)
        }

  var concepttypes = myDoc.getElementsByTagName("ConceptType");
  var aantal_concepttypes = concepttypes.length;

  for (n=0; n<aantal_concepttypes ; n++){
        document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

        document.write("<tr><td width=\"60\%\"><b>Concept Type : </b></td><td
width=\"40\%\"><b> " + concepttypes(n).attributes(0).nodeValue + "</b></td></tr>");


        var objecttype = concepttypes(n).getElementsByTagName("ObjectType");
        for (i=0 ; i<objecttype.length ;i++)
                {
                document.writeln ("<tr><td width=\"60\%\"> Object Type : </td>" + "<td
width=\"40\%\">" + objecttype(0).attributes(0).nodeValue + "</td></tr>");

                var with_roles = objecttype(0).getElementsByTagName("with");
                for (i=0 ; i<with_roles.length ;i++)
                        {
                        document.writeln("<tr><td width=\"60\%\" align=\"right\">Object
plays role:___ </td>" + "<td width=\"40\%\">"
                                                + with_roles(i).attributes(0).nodeValue +
"</td></tr>");
                        }


                }
        var relationtype = concepttypes(n).getElementsByTagName("RelationType");
        for (i=0 ; i<relationtype.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> Relation Type : </td>" + "<td
width=\"40\%\">" + relationtype(0).attributes(0).nodeValue + "</td></tr>");

                var has_roles = relationtype(0).getElementsByTagName("has");
                for (i=0 ; i<has_roles.length ;i++)
                        {
                        document.writeln("<tr><td width=\"60\%\"
align=\"right\">Relation has role:___ </td>" + "<td width=\"40\%\">"
                                                + has_roles(i).attributes(0).nodeValue +
"</td></tr>");
                        }


                }

        var overviewed_on = concepttypes(n).getElementsByTagName("overviewed_on");
        for (i=0 ; i<overviewed_on.length ;i++)
                {
```

```
                document.writeln("<tr><td width=\"60\%\"> Overviewed on prototype page
: </td>" + "<td width=\"40\%\">" + overviewed_on(i).attributes(0).nodeValue +
"</td></tr>");
                }
        var with_overview_on =
concepttypes(n).getElementsByTagName("with_overview_on");
        for (i=0 ; i<with_overview_on.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> With overview on page  :
</td>" + "<td width=\"40\%\">" + with_overview_on(i).attributes(0).nodeValue +
"</td></tr>");
                }

        }
  document.write("</table>");

}



concepttypes()
</SCRIPT>
```

```
<a href="schema1.htm" ><font
        size="2">Back to menu</font></a>
</body>
</HTML>
```

Example 4

```
<HTML>
<HEAD>
<TITLE> Presenting the Pages of Websites </TITLE>
</HEAD>
<body bgcolor="#BCD3E">
<XML ID="xmldoc" ></XML>
<h3> Presenting the pages of the Y2K website</h3>
<SCRIPT>

function load() {
  xmldoc.load("website.xml");

  }

function pages() {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
        {
        alert(myDoc.parseError.reason)
        }

  var pages = myDoc.getElementsByTagName("Page");
  var aantal_pages = pages.length;

  for (n=0; n<aantal_pages ; n++){
        document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

        document.write("<tr><td width=\"60\%\"><b>Page : </b></td><td
width=\"40\%\"><b> " + pages(n).attributes(0).nodeValue + "</b></td></tr>");

        var pageinstance = pages(n).getElementsByTagName("PageInstance");
        for (i=0 ; i<pageinstance.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> PageInstance : </td>" + "<td
width=\"40\%\">" + pageinstance(0).attributes(0).nodeValue + "</td></tr>");

                var has_page_prototype =
pageinstance(0).getElementsByTagName("has_page_prototype");
                for (i=0 ; i<has_page_prototype.length ;i++)
                        {
                        document.writeln("<tr><td width=\"60\%\" align=\"right\">Page
has page prototype:__ </td>" + "<td width=\"40\%\">"
                                                +
has_page_prototype(0).attributes(0).nodeValue + "</td></tr>");
                        }
                }

        var contains = pages(n).getElementsByTagName("contains");
        for (i=0 ; i< contains.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> this page contains the
concept  : </td>" + "<td width=\"40\%\">"
                                                                                +
contains(i).attributes(0).nodeValue + "</td></tr>");
                }

        var overview_page_of = pages(n).getElementsByTagName("overview_page_of");
        for (i=0 ; i<overview_page_of.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> This page provide an overview
for  : </td>" + "<td width=\"40\%\">" + overview_page_of(0).attributes(0).nodeValue +
"</td></tr>");
                }

        var is_source_for = pages(n).getElementsByTagName("is_source_for");
        for (i=0 ; i<is_source_for.length ;i++)
                {
```

```
               document.writeln("<tr><td width=\"60\%\"> This page is the source for
link  : </td>" + "<td width=\"40\%\">" + is_source_for(0).attributes(0).nodeValue +
"</td></tr>");
               }

       var is_target_for = pages(n).getElementsByTagName("is_target_for");
       for (i=0 ; i<is_target_for.length ;i++)
               {
               document.writeln("<tr><td width=\"60\%\"> This page is the target for
link  : </td>" + "<td width=\"40\%\">" + is_target_for(0).attributes(0).nodeValue +
"</td></tr>");
               }

       var is_source_for_proto = pages(n).getElementsByTagName("is_source_for_proto");
       for (i=0 ; i<is_source_for_proto.length ;i++)
               {
               document.writeln("<tr><td width=\"60\%\"> This page is the source for
link prototype  : </td>" + "<td width=\"40\%\">" +
is_source_for_proto(0).attributes(0).nodeValue + "</td></tr>");
               }

       var is_target_for_proto = pages(n).getElementsByTagName("is_target_for_proto");
       for (i=0 ; i<is_target_for_proto.length ;i++)
               {
               document.writeln("<tr><td width=\"60\%\"> This page is the target for
link prototype : </td>" + "<td width=\"40\%\">" +
is_target_for_proto(0).attributes(0).nodeValue + "</td></tr>");
               }

       }
  document.write("</table>");

}



pages()
</SCRIPT>
```

```
<a href="schema1.htm" ><font
        size="2">Back to menu</font></a>
</body>
</HTML>
```

Example 5

```
<HTML>
<HEAD>
<TITLE> Presenting the Links of Websites </TITLE>
</HEAD>
<body bgcolor="#BCD3E">
<XML ID="xmldoc" ></XML>
<h3> Presenting the Links of the Y2K website</h3>
<SCRIPT>

function load() {
  xmldoc.load("website.xml");

  }

function Links() {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
        {
        alert(myDoc.parseError.reason)
        }
  var a="Link"
  var links = myDoc.getElementsByTagName(a);
  var aantal_links = links.length;

  for (n=0; n<aantal_links ; n++){
        document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

        document.write("<tr><td width=\"60\%\"><b>Link : </b></td><td
width=\"40\%\"><b> " + links(n).attributes(0).nodeValue + "</b></td></tr>");

        var linkinstance = links(n).getElementsByTagName("LinkInstance");
        for (i=0 ; i<linkinstance.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> LinkInstance : </td>" + "<td
width=\"40\%\">" + linkinstance(0).attributes(0).nodeValue + "</td></tr>");

                var has_link_prototype =
linkinstance(0).getElementsByTagName("has_link_prototype");
                for (i=0 ; i<has_link_prototype.length ;i++)
                        {
                        document.writeln("<tr><td width=\"60\%\" align=\"right\">Link
has link prototype:__ </td>" + "<td width=\"40\%\">"
                                                +
has_link_prototype(0).attributes(0).nodeValue + "</td></tr>");
                        }
                }

        var from = links(n).getElementsByTagName("from");
        for (i=0 ; i< from.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\" >Link starts from page   :
</td>" + "<td width=\"40\%\">"
                                                                                +
from(0).attributes(0).nodeValue + "</td></tr>");
                }

        var to = links(n).getElementsByTagName("to");
        for (i=0 ; i<to.length ;i++)
                {
                document.writeln("<tr><td width=\"60\%\"> Link goes to page  : </td>" +
"<td width=\"40\%\">"
                                                                                +
to(0).attributes(0).nodeValue + "</td></tr>");
                }
        }
  document.write("</table>");

}
```

```
Links()
</SCRIPT>

<a href="schema1.htm" ><font
        size="2">Back to menu</font></a>
</body>
</HTML>
```

Example 6

```
<HTML>
<HEAD>
<TITLE> Presenting the relations in Websites </TITLE>
</HEAD>
<body bgcolor="#BCD3E">
<XML ID="xmldoc" ></XML>
<h3> Presenting the relation of the Y2K website</h3>



<SCRIPT>
function load() {
  xmldoc.load("website.xml");

  }
```

```
function relationtype(relation) {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
       {
       alert(myDoc.parseError.reason)
       }


  var relationtypes = myDoc.getElementsByTagName("RelationType");

  document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

       for (l=0 ; l<relationtypes.length ;l++){
              if ( relationtypes(l).attributes(0).text == relation ){
```

```
              document.writeln("<tr><td width=\"60\%\"> Relation Type : </td>" + "<td
width=\"40\%\">"
                                                      +
relationtypes(l).attributes(0).nodeValue + "</td></tr>");
              var has_roles = relationtypes(l).getElementsByTagName("has");
              var roles = myDoc.getElementsByTagName("Role");

              for (m=0 ; m<has_roles.length ;m++){
                     for (n=0 ; n<roles.length ;n++){
                     if (has_roles(m).attributes(0).nodeValue ==
roles(n).attributes(0).nodeValue){

                     var role_ = roles(n).attributes(0).nodeValue;
                     var played_by = roles(n).getElementsByTagName("played_by");
                     for (o=0 ; o<played_by.length ;o++)
                     {
```

```
                            document.writeln ("<tr><td width=\"60\%\"  > Object
</td>" +
                                                "<td width=\"40\%\">" +
played_by(0).attributes(0).nodeValue + "</td></tr>");
                     }

                     document.writeln ("<tr><td width=\"60\%\" align=\"right\" >
Object plays role : </td>" +
                                                "<td width=\"40\%\">" + role_ +
"</td></tr>");

              }
              }}
              }}
  document.write("</table>");
```

```
}
```

```
function objecttype() {
  load()
  var myDoc=xmldoc;
  if (myDoc.parseError.reason !="")
        {
        alert(myDoc.parseError.reason)
        }
        var object = prompt ("Of what Object do you want to see the relations?");
        var objecttypes = myDoc.getElementsByTagName("ObjectType");


        document.write("<table border=1 width=\"100%\" bgcolor=\"#BCcccc\"><br>");

        for (i=0 ; i<objecttypes.length ;i++){
                if ( objecttypes(i).attributes(0).text == object ){


                document.writeln("<tr><td width=\"60\%\"> ObjectType : </td>" + "<td
width=\"40\%\">"
                                                                +
objecttypes(i).attributes(0).nodeValue + "</td></tr>");
                document.write("</table>");

                var has_roles = objecttypes(i).getElementsByTagName("with");
                var roles = myDoc.getElementsByTagName("Role");

                for (j=0 ; j<has_roles.length ;j++){
                        for (k=0 ; k<roles.length ;k++){
                        if (has_roles(j).attributes(0).nodeValue ==
roles(k).attributes(0).nodeValue)
                        {
                        var relation = roles(k).getElementsByTagName("in");
                        var relation_ = relation(0).attributes(0).nodeValue;
                        relationtype(relation_);

                        }
                     }
                    }
}}}
objecttype()

</SCRIPT>




<a href="schema1.htm" ><font size="2">Back to menu</font></a>
</body>
</HTML>
```

References

[BD 98] Fank Boumphrey, Olivia Direnzo, …
Programmer to Programmer XML Applications
Wrox Press Ltd. 1998 ®

[DM 96] Lois M. L. Delcambre, David Maier, Radhika Reddy, Lougie Anderson
Structured Maps: modeling explicit semantics over a universe of information
Digital Libraries © Springer-verslag 1997

[GP 98] Charles F.Goldfarb, Paul Prescod
The XML Handbook
© 1998 Prentice Hall PTR.

[HA 95] Terry Halpin
Conceptual Schema & Relational Database Design (Second Edition)
© 1995 Prentice Hall Australia Pty Ltd

[NN 96] Netscape Navigator (version4.0)
JavaScript Guide
©Netscape Communications Corporation 1996

[TB 97] Timothy Budd
An introduction to Object-Oriented Programming (Second Edition)
© 1997 Addison-Wesley Longman,Inc.

[WK 96] Jos Warmer & Anneke Kleppe
Praktisch OMT
© 1996 Addison-Wesley Nederland BV.

http://w3c.org
http://msdn.microsoft.com/xml/
http://www.xml.com/

http://lislin.gws.uky.edu/Sitemap/Sitemap.html