Product Document Management with SGML and Relational Databases

Heikki Toivonen

Master of Science Thesis

20.4.2000

University of Jyväskylä
Department of Mathematical Information Technology

Tiivistelmä

Tietokannat ja rakenteiset dokumentit perustuvat niin erilaiseen teknologiaan ja ajatteluun, että niiden yhteiskäyttö voi olla ongelmallista. Edistystä on kuitenkin tapahtunut. Tämä tutkielma käsittelee näiden kahden teknologian eroja sekä vaikeuksia teknologioiden yhteiskäytössä. Pääpaino on tuotedokumentaation hallinnassa. Käytännön osuudessa esitellään eräs sovellus, jossa tietokannat ja rakenteiset dokumentit tukevat toisiaan.

Tekijä: Heikki Toivonen

Yhteystiedot: sähköposti `hjtoi@jyu.fi`

Työn nimi: Tuotedokumentaation hallinta SGML:n ja relaatiotietokantojen avulla

Avainsanat: SGML, XML, HyTime, rakenteiset dokumentit, dokumenttien hallinta, tuotetiedon hallinta, tietokannat

Abstract

Databases and structured documents have been used apart from each other. The situation has changed dramatically over the past few years. This thesis discusses differencies and difficulties in making the two separate realms interact. The emphasis is on product document management. The practical part of the thesis shows one implementation where databases and structured documents work together.

Author: Heikki Toivonen

Contact: Email `hjtoi@jyu.fi`

Title: Product Document Management with SGML and Relational Databases

Keywords: SGML, XML, HyTime, structured documents, document management, product data management, databases

Acknowledgements

This thesis was years in the making. Partly the reason was that I got tired of writing it, but also because my job was taking too much time. In a way that was a good thing, because it enabled me to gain more experience and make this a better paper. It was also interesting to see new standards and programs emerging. I was also able to look back with a better hindsight of what should have been done differently, and I assume I took a more critical view of my own work.

I would like to thank the various persons who have inspired me on my way of learning SGML and helped either directly or indirectly with this thesis: the authors of Mosaic, for introducing me to structured document principles with the birth of the World Wide Web; Professor Heather Brown from the University of Kent at Canterbury for teaching me SGML and supervising my first SGML project during my time at Kent University; Ralf Petell and Björn Peltonen for offering me work at CiTEC Engineering Oy; Joakim Östman for project management; Leonard Norrgård for help with programming; Kaisa Miettinen for getting gears rolling when they got stuck; Michael Leventhal for language checking and other comments; Carl-Johann Måsala of Wärtsilä for checking Wärtsilä facts; Jukka-Pekka Santanen and Pasi Koikkalainen, my thesis supervisors at the University of Jyväskylä, and all the rest of the supercool SGML gang at CiTEC for the enjoyable and interesting times. Many people have written great books and other publications, their names can be found in References. I must still point out that all errors are mine.

The people at the 4th International HyTime Conference held in Montreal, Canada, deserve a special mention. That was the first SGML/HyTime meeting I attended, and the presentations and conversations proved to be very inspiring and helpful to me. The atmosphere was enthusiastic and friendly, even towards a newcomer like me. And to actually meet and speak with creators of SGML and HyTime was more than I could have ever dreamed of — talk about motivation!

Last but not least I would like to thank my wife Virpi for her understanding and patience.

Vaasa

Heikki Toivonen

Table of Contents

List of Figures

List of Examples

Terms And Acronyms

Quotes are from the HyTime standard [ISO97].

anchor

> "An object (or a list of objects) that is linked to other objects or lists of objects by a hyperlink." Object is not a formal construct in HyTime, but it can mean a document, an element in a document, a rectangular area in a frame in a video sequence or just about anything.

attribute

> SGML and XML **elements** may contain attributes. Attributes contain **metadata** of the **element**.

catalog

> Catalog files map **public identifiers** to **system identifiers**. They are plain text files.

contextual hyperlink

> "A hyperlink that occurs 'in context', meaning that one anchor of the link is the link element itself [...] and is a traversal initiation anchor."

document type definition

> SGML and XML document structure specification is called document type definition (**DTD**), which is the description of the structure of an **SGML** or **XML** document written in a formal language.

DSN

> Data Source Name is a concept from **ODBC**. It is possible to define a DSN for a database and access the database with that name, without knowing the actual location of the database. This is possible because the ODBC layers take care of that information.

DTD

> Abbreviation for **document type definition**.

element

> SGML documents consist of elements that contain other elements and text. Start **tags** begin elements and end **tags** close elements.

entity

> SGML has several kinds of entities. Parameter entities are used inside a **document type definition** to reuse DTD constructs. Internal entities can be used in DTDs and document instances, and they can expand to text and **markup**. External entities refer to external text,

markup or other objects like images.

grove

“Graph Representation Of property ValuEs.” A grove is the parse tree that a parser produces in memory. It also contains some additional information, like links between the nodes in the tree (or forest, which consists of multiple trees).

hyperlink

“An information structure that represents a relationship among two or more objects.”

hypertext

“Information that can be accessed in more than one order.”

HyTime

Hypermedia/Time-based Structuring Language. HyTime is an international standard for representing multimedia documents, including links. HyTime uses SGML constructs (all HyTime documents are legal SGML documents, but one needs a HyTime-aware processor to understand the HyTime semantics).

link

For the purposes of this thesis a link is the same as **hyperlink**.

markup

In the case of structured documents, the document structure is specified with markup. The markup is part of the **metadata** of the document.

metadata

Data about data, for example the creation date of a file is metadata about the file.

micro-document

A document, often small. Usually micro-documents are assembled to create complete manuals.

Multidoc Pro

An **SGML** browser by Citec Software Ltd Oy.

navigator

A **Multidoc Pro** term for an electronic table of contents.

notation

Entities can refer to external objects of different type. For example, images can be in GIF and JPEG format. A notation can be used to define these types.

ODBC

Open Database Connectivity. A Microsoft standard through which applications can access different databases. ODBC drivers hide the actual differences between database implementations.

public identifier

Certain SGML and XML objects can be referred to by public identifiers. These include **document type definitions**, **entities** and **notations**.

reference concrete syntax

The SGML standard defines a default SGML declaration that is assumed if no explicit SGML declaration is given. This is called the reference concrete syntax.

SGML

Standard Generalized Markup Language. SGML is an international standard for structured documents. Explicit structure in documents helps manage them and enables computer programs to work intelligently with the structure.

SGML declaration

The first part of an SGML document that specifies things like character encoding and the maximum length of names which may be used in the document **markup**.

SGML (document) instance

The third major part of an SGML document, the “actual document“.

stylesheet

A stylesheet describes how a document should be formatted. Structural documents usually separate the content of the document and the style information into different files.

system identifier

A SGML **public identifier** is first mapped to a system identifier, which is used by the system to locate the physical object, for example a **document type definition**. System identifir is usually a file name.

tag

The **markup** in the **SGML document instance** consists of, among other things, tags that mark the boundaries of **elements**. A start tag begins an **element**. It is possible to give values to the element’s **attributes** inside the start tag. The **SGML declaration** defines what characters are used to begin and end tags. The **reference concrete syntax** specifies that `<` opens a start tag, `</` opens an end tag and `>` closes a tag. This is a sample start tag: `<title lang=”en”>`. End tags close elements, and they cannot have attributes. The end tag for the start tag would look like this: `</title>`.

web

A web in **Multidoc Pro** term which refers to a file that can be loaded over an existing SGML document and which may contain user defined links, annotations and bookmarks for that document.

XML

Extensible Markup Language. XML is a World Wide Web Consortium Recommendation (effectively an Internet standard). XML is a simplified subset of SGML.

Chapter 1

Introduction

Everything has structure.

Unknown

The driving force for this paper was the Product Information Management project started at Wärtsilä NSD Power Plants (in Vaasa, Finland) to overcome problems in product documentation. The solutions presented in this paper were developed mostly in 1996 and 1997.

This thesis describes how product documents in structural format can be managed effectively with relational databases. The structural document format this thesis deals with is Structured Generalized Markup Language (SGML), which is an International Standards Organization (ISO) standard [ISO86].

SGML has also evolved for the World Wide Web (WWW). This format of SGML is known as the Extensible Markup Language (XML) and it is a World Wide Web Consortium (W3C) Recommendation [W3C98]. Prior knowledge of SGML nor XML will not be needed to understand this thesis as they will be explained with sufficient details to understand this document. Relational databases and Product Data Management (PDM) will also be described briefly for the same reason. From the point of view of this thesis SGML and XML are nearly identical. Differences will be clearly pointed out where they matter. This thesis generally refers to SGML as the structural format, but generally XML could be used equally well.

This thesis will show the importance of effective management of product documents. The advantages of standardized, structured format will be shown to be superior to traditional approaches. The management of such documents is equally important. It will be explained why relational databases, while not the most advanced technology available today, are still good for the job. Moreover it will be shown that it is feasible to retrieve information from both databases and structured sources and combine the information so that it can be shown as a structured document. Methods for using databases to manage SGML and XML documents and assemble large documents from document fragments will be discussed as well.

This document is divided into two parts. The first part explains the different standards and technologies used and discusses the theory behind the practical part of this thesis. The first Part is divided into five chapters. Chapter 2 describes product data management. Chapter 3 describes structured documents. Chapter 4 explains databases and the last chapter in the first part discusses the theory of managing documents with databases.

The second part is divided into four chapters and it describes the SGML document management system at Wärtsilä NSD. Chapter 6 gives an overview of the system. Chapter 7 describes the document authoring process and Chapter 8 how the different documents are managed and assembled into larger units.

The implementation of the tools described in the second part happened mostly in 1996 and 1997. Some software available at that time has disappeared from the market while new products have appeared in their place. Great advances in standards and other technologies have occurred. The implemented technologies will be discussed in light of new information.

Each chapter begins with a quote, often from a science-fiction novel. The quote is somehow related to the chapter in question. The actual relationship is left as an exercise to the reader.

Some images (screenshots particularly) are not of the highest quality. That is because the original image was in some non-vector format like GIF that suffers from scaling. Scaling was needed in many places to make the images fit on paper.

This document itself is in structured format. The first drafts were written with Adept•Editor[1]. Additional work was done with various other SGML editors including FrameMaker+SGML[2], which was also used for printing and conversion to various other formats. Occasionally text was edited with plaintext editors. The Document Type Definition (DTD) was the `Docbook` DTD [OAS99], which was slightly modified for the needs of this thesis. The original structured format was SGML, but simple transformations were made to create an XML version as well.

There are Synex ViewPort[3] stylesheets as well as Cascading Style Sheets (CSS) [W3C96] for this document, which allows viewing of this document in any SGML or XML-capable browser that supports either of the two stylesheet formats. Additionally, PostScript and PDF versions of this

1. Adept•Editor is a product of ArborText Inc.
2. FrameMaker+SGML is a product of Adobe Systems Inc.
3. Synex ViewPort is a product of Synex Information AB, a fully owned subsidiary of Enigma, Inc. See `http://www.synex.se`.

thesis were extracted for final printing. An HTML version was also produced to allow viewing of this document with less advanced Web browsers.

Some words about the SGML markup used in this document is in order. The first occurrence of an important term is wrapped in `FirstTerm` element and is formatted in **bold face** on paper. Code and SGML markup examples may be nested inside one of several different elements, but it will always be formatted with fixed pitch `Courier font`. The term `element` will be explained later. Most of the acronyms and terms are also explained in Terms and Acronyms. Longer code and markup examples are numbered, as shown below:

Example 1: Markup Sample.

```
<tag>Some SGML/XML sample</tag>
```

Chapter 2

Product Data Management

I don’t understand, came Briareus’s code on the common band. *It opened to nowhere*.

Not to nowhere, sent Nemes, reeling in the filament. *Just to nowhere in the old Web. Nowhere the Core has built a farcaster.*

That’s impossible, sent Scylla. *There are no farcasters except for those the Core has built.*

Nemes sighed. Her siblings were idiots. *Shut up and return to the dropship*, she sent. *We have to report this in person. Councillor Albedo will want to download personally.*

Dan Simmons, *The Rise of Endymion*

Product Data Management (PDM) is a way to manage people and other resources and product development processes. It is crucial to most companies and organizations. As such it is no wonder there are several opinions about it and lots of material about it. Good overall guides to the subject are, for example [CIM98] and [PDM97a]. These works were used as source material for this chapter. PDM has also been the target of extensive standardization work. The most well-known standard for product data is, without question, the Product Data Representation and Exchange (STEP) [ISO94].

The practical part of this thesis evolved because of a need for better management of documents. To understand document management some knowledge of the more general problem of Product Data Management is needed. This chapter will give a brief introduction to these concepts or the relevant concepts in this area. In addition, product document management is introduced and the justification for its importance is pointed out.

2.1 What Is Product Data Management?

Product Data Management can be thought of as the umbrella word covering (among other things) Engineering Data Management (EDM), document management, Product Information Management (PIM) and technical data management. It is being used anywhere there is some kind of a product being manufactured, sold or maintained. The word product should be understood very widely. A product can be anything from an airplane to a computer program or a service. “The Challenge is to maximize the time-to-market benefits of concurrent engineering while maintaining control of your data and distributing it automatically to the people who need it - when they need it” [PDM97a].

PDM evolved from systems built in-house into commercial systems in the 1980s. The vendors in those days were already involved with CAD or other computer aided engineering systems. Initially the systems were mostly concerned with just engineering data, but recently support for the whole product life cycle has improved as well [PDM97b].

A PDM system consists of a data repository or vault (often a relational database), a set of user functions and a set of utility functions. A PDM system’s commands are either embedded into other applications (CAD, word processors and so on) or the commands from those other systems are embedded into the PDM system.

The PDM system stores two kinds of data: product data and **metadata**. Metadata is data about data, and in the case of a PDM system it helps the PDM system carry out the tasks it is supposed to do. The distinction between data and metadata is a bit vague. One application’s data can be metadata for another application.

The user functions in a PDM system can be divided into five categories: document management, workflow and process management, product structure management, classification and, finally, program management. The PDM system must securely and effectively manage and protect the data from both accidental or deliberate attempts to destroy data integrity. Logged access control keeps track of who has got what information and when they have obtained it. Access control can also prevent unauthorized access to data. Release management makes sure the data goes through the required steps before ending up at the customer (for an example, see Figure 1). Document management (or version control) can take care of logging and managing ad hoc demands to data.

Workflow and process management can make the system proactive. Predefined paths for data can be set up for repetitive tasks, which not only ensures the correctness of the overall task, but also improves efficiency. This is very closely related to the conveyor belts in a factory. Product structure management covers bills of materials as well as product configurations. Bills of materials can even be created automatically from product structures. As all information is gathered in a central place, it is easy to see what things will be affected by changes. Finding the information in the first place is easy as well. The PDM system can show different views of the data, such as structural relationships, documentation and support information. Classification allows grouping of similar parts and information, which in turn can cut the time spent in re-design. Finally, program management completes the user functions. Program management tools provide work breakdown structures to make it easier to arrange resources, predict schedules and, in general, track how projects are doing.

Customization is needed to better fit an off-the-self PDM system to an organization’s needs. The utility functions offer a way to do this without reprogramming the whole PDM system. The utility functions are also needed to support the user functions described above. Typical utility functions include communication and notification services, data transport, data translation and system administration as well as customized reports for different user groups and purposes.

Communication is improved simply by using a PDM system because everyone using the system has access to all the information in the system (access control restrictions apply, of course) in real time. The PDM system can automatically send notification emails about unexpected delays or otherwise important happenings. The system can automatically transfer data from its storage to the user, so the user need not know where it is actually located. Even in a moderately large organization not everyone can work with the same exact word processors and drawing programs so automatic data translation or transformation can be set up in the PDM system. Administration functionality offers utilities to change access control, workflows and data-backup.

As can be seen from the wide area of functionality covered, a PDM system must be designed in such a way that third-party components can be readily integrated into the main system. The default user interface and terminology used in the PDM system may also be customizable for the specific or unique needs of each organization using the system.

That pretty much covers the basics of product data management. As this thesis is mostly concerned with documents and documentation, lets take a more thorough look at documentation and document management.

2.2 There Is No Product without Documentation

Simply put, there are no products that do not have documentation associated with them.

Let us think of a nail: A straight thin piece of iron or some other material, sharp at one end and flattened at other. One would imagine a product cannot be much simpler than that. But we shall take a closer look.

Let us say we would like to make a new nail. First we must determine where it should be used. This requires some thought and maybe research, which will produce our first documents. Suppose there was an area that would benefit of a new type of a nail. Then we must inspect the requirements, which will require additional documents. These documents could say that the nail must be at least three inches in length but no more than four inches, it must be able to withstand corrosive substances and it must be very thin. We would then manufacture some prototypes that would go into testing. We would get test reports. By the time the nail is perfect in its design we would have a big pile of paper (or at least lots of files on a computer). The nail would then be manufactured in large quantities, the factory might need to be customized for it, some subcontractors would probably be required to deliver the alloys and so on. By the time the nail reaches retail shops there would be a mountain of information about it. And it would still go on: customers would give feedback, there would be sales figures and so on. Today one just cannot live without documents.

Given this simple example with the nail it is remarkable that product documentation is usually in very bad shape. In many cases documentation is seen as the least important piece of the whole product. The process to create a user manual for a product often does not start until the product is ready to be shipped to customers. This may lead to delays in the product release. A very bad situation indeed. A nail would not have too big a user manual, but a cellular phone or diving gear would be almost useless without some kind of instructions.

A document is as important part of a product as any other tangible thing. Maybe even more so. In this information society knowledge is regarded as the most precious commodity. In some cases the document itself is the product.

The same rules that govern the management of other products apply to document management as well. Typical life of a document is presented in Figure 1. The planning for a new document usually begins when a new product is being planned, or a need for a certain kind of a document is discovered. New legislation or company policies often result in new documents. Authors or technical writers create the actual document content with the help of other documents or experts. The authoring work is often easy to outsource.

The document then goes through the checking phase where facts, spelling, readability and other things are checked. It can be returned back to the author (with more documentation explaining what was wrong) or it can advance through the approval process. The persons that approve documents often take the legal responsibility that the document is correct. If the document is a user manual for a machine that can potentially be dangerous to humans this is a big responsibility indeed. Once the approval stamp is on a document it is considered “frozen” and it is released, possibly with other products. User comments and new versions of products can trigger changes to the documentation which end up in new “frozen” revisions of the original document.

Check-in, check-out and locking are important concepts in document authoring. A document database handles these tasks and the database holds the single up-to-date copy of any given document. When a user wants to edit a document, he must first check out a work copy from the document database. The database marks the document as locked and will only allow reading of the locked document. Once the user has finished editing, he must check the document back in. This clears the lock. Often, only the differences between document versions are stored to save disk space. Some systems have more advanced locking models; for example a document can have multiple locks, some locks can deny reads and so on.

Documents can be prepared as if they were being built on an assembly line. The planning phase may create some skeleton documents and additional guidelines for the document. Multiple authors can add to and refine the skeleton document one after the other until it is finished. An automated workflow system can improve productivity immensely, because moving a “job” from one person to the next is instantaneous. Because the workflow has been etched into the computer system the “job” will always move to the person it is supposed to go to and not to somebody else. In some large corporations files can sit for days waiting to be moved from one person to another.

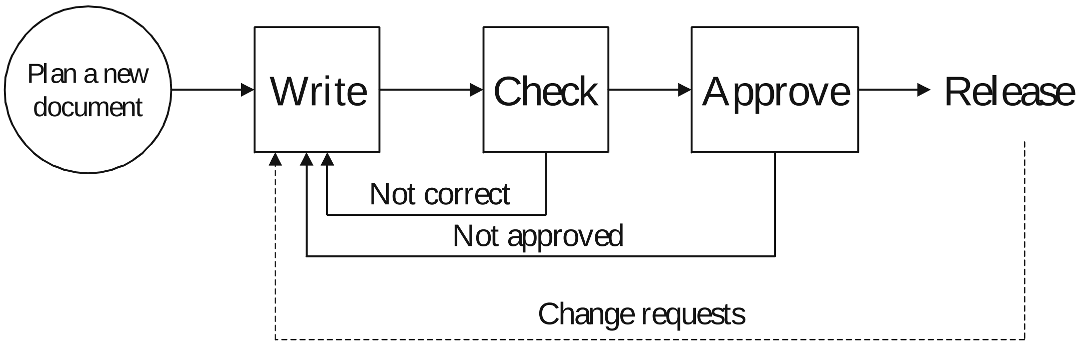


Figure 1: Document Life Cycle.

If a document database is being used, documents may also be edited simultaneously by multiple authors. Document databases that are specialized for structured documents can easily lock parts of documents while allowing edits in other parts. Normal version control systems can handle simultaneous edits if the documents are not in a binary format.

Let us consider an example where two authors are editing the same document at the same time. The first author to check his changes in does not notice anything unusual, but the second author will notice that the version control system reports that his version does not match what is in the version control system and merging is required. In some cases the system can do an automatic merge while in other cases manual work is needed. The situation with two authors editing the same document with the help of a version control system is presented in Figure 2. The boxes labeled `Doc <number>` show the document stored in the version control system, and the number refers to the version number in the version control system.

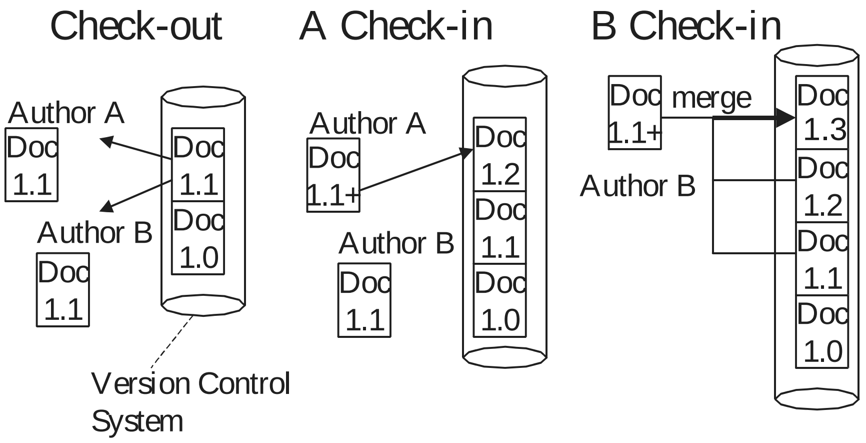


Figure 2: Check-out, Check-in and Merge with Version Control System.

Next we will have a look of the economic issues concerning documents and documentation.

2.3 Economic Issues

The amount of time and money spent on producing new information is staggering. For example, 20% of the GNP of the United States is spent on generating new information. Over 90% of the information is in documents [Arb95]. The amount of electronic documentation is growing 20-60% each year while the same figure is 10% for paper documentation (in the USA) [Onn99].

Here is a specific example having to do with oil rigs. It is estimated that about half of the manufacturing costs is in documentation [Pel97b][1]. A recent newspaper mentioned that the cost of setting up a new oil rig in the Norwegian waters would cost over 11 billion Finnish marks[2]. Even a moderate 10% saving in documentation would therefore save nearly 600 million Finnish marks from the total costs of such an oil rig.

The World Wide Web is also growing rapidly. The amount of text in English grows about 50% per year. The figure for non-English pages is 90%. This will also lead to an increasing need for translations [Kla98].

It has been estimated that authors spend up to 30% of their time searching for information and roughly the same amount of time laying out the text to produce nice printouts [Arb95]. SGML addresses both aforementioned areas of authoring. First, because of SGML’s structured nature, documents conforming to this standard can be archived and searched much more efficiently than,

1. This was also mentioned in the presentation “Toward STEP Interchange: Seeing the Document as a Snapshot of the Data” given by Daniel Rivers-Moore at the 4th International HyTime Conference in Montreal, Canada.
2. This was mentioned in an article in the Finnish “Kauppalehti” during winter 1999. 11 billion Finnish marks is roughly 2 billion US dollars.

say, Microsoft Word documents. Secondly, authors do not have to spend their time worrying about the layout. This is often unproductive use of time as the publisher will want to change the layout to conform to their standards. Laying out and publishing an SGML document can be completely separated from the authoring process.

Technical manuals are often huge, while usually only a small part of the whole document is needed. The nature of SGML makes it easy to extract tailored subdocuments from large documents. SGML databases make it possible to have multiple authors working on the same document at the same time, because each author needs to lock only the small piece he is currently working on.

Other important areas not yet mentioned include document interchange and long term storage. While standard word processors emerge with a new proprietary save format for every major update, SGML has been around since 1986, and is likely to remain usable for decades — that is why the term “Everlasting Information” is sometimes used with documents conforming to the SGML standard. Without long lived, standard document formats our electronic era can leave a black hole in information to future historians because it may simply be impossible to find both the software and hardware to read some exotic file format in the future. Already it is nearly impossible to read the data from early magnetic tapes and punched cards! With SGML only DTDs evolve, and changing document instances so that they conform to newer versions of DTDs is generally a lot easier than changing some word processor's format. Different versions of DTDs can also coexist, so conversion in many cases is not even needed. In addition, SGML is platform independent i.e. as long as ASCII files can be transferred from one system to another can SGML files be interchanged. Archival issues have been explored, for example, in [Onn99] and [Met99].

This section has developed the claim that SGML can make documents and the processes needed in documentation more effective. The following chapters will show how and why is this possible.

Chapter 3

Structured Documents

HyTime is the borg standard.

W. Eliot Kimber

All the information we have gathered into something that could be considered to be a document has structure. Sometimes the structure is very explicit, like in a computer program, so that even computers can understand them. Other documents have only implicit structure, and we may not even know or recognize that something has structure. For a human being a computer program represented as 0's and 1's would be meaningless, and devoid of structure. On the other hand, most computer programs would not make anything out of this paragraph. The information in this chapter was collected mostly from [Tra95], [Gol90], [W3C98], [DeR94] and [Kim98].

3.1 Ways to Indicate Structure

All documents have at least implicit structure. For some documents the structure has been declared explicitly, either internally or externally. If the structural information is internally contained in the document, it is usually called markup.

LaTeX [Lam94] is an example of an internal document markup language; where the structural and layout information is mixed with the content. The beginning of a section, in the source file, looks something like this:

Example 2: LaTeX Sample.

```
\section{Onion Pie}
\begin{list}
\item large onions
\item large tomato ...
```

The example above would be formatted by a LaTeX system so that a section heading would appear first, followed by a bulleted list of ingredients for the onion pie. How the heading and list would appear can be defined (although this is rather difficult), but the default heading would look pretty much the same as the section headings in this thesis. All the markup (like `\begin{list}`) would be processed by the software and it would not end up in the actual layout.

An example of external markup is the graphics format for Regenesis[1], the first graphical multi-user dungeon (MUD). The graphics are simple colored polygons that consist of a maximum of 32 vertices. There can be a maximum of 32 polygons in a scene. Regenesis uses 16 colors. The coordinate system is a 256 by 256 matrix. An image is represented as a string of hexadecimal numbers. As such it has no apparent structure. But that string has structure when it is viewed with the external information: the programs drawing the images must of course know what to do with the string! The first two characters from the string, when converted to a hexadecimal number, is the number of polygons in the string, the next number is the number of vertices in the first polygon and so on. See Figure 3 for sample.

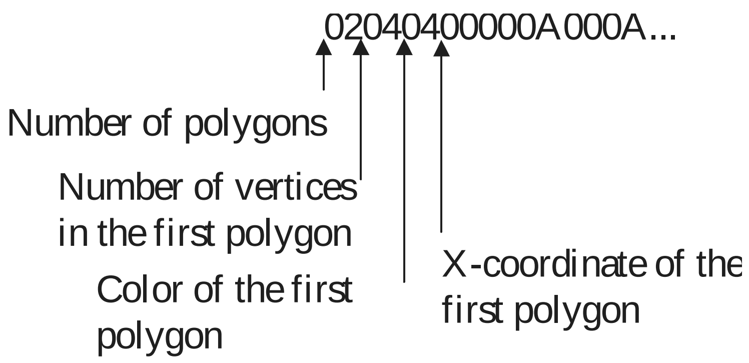

02040400000A000A...
Number of polygons
Number of vertices in the first polygon
Color of the first polygon
X-coordinate of the first polygon

Figure 3: The Structure of Scene Markup in Regenesis.

There is a hybrid version of the internal/external markup. Some documents contain the structural

1. Regenesis may have been the first online game with graphics. Its "children" are Doom, Quake and Warbird, for some examples. There is a WWW page describing the MUD at `http://www.lysator.liu.se/mud/bsxmud.html`. To actually play the game one needs a client program that contacts the MUD server. Regenesis was originally written by Bram Stolk.

markup in the beginning or in the end of the document, which contains pointers to the actual content. Microsoft Word uses this technique [Tra95].

Structured documents are normally formatted to get usable view of the data. It would not make much sense to try to read the binary representation of a computer program, for example. Running the program presents the structured data in an understandable format.

3.2 Languages And Parsers

Structured documents use some language to describe the structure. For example, the C++ programming language is standardized and there is software that can do meaningful things with data in C++ notation.

All languages (or more precisely, grammars) can be divided into several different categories, for example regular and context free languages. Any book on the theory of computation or compiler design will discuss the theory of languages, for example [Sip96]. There is no need to understand language theory in depth, for the purposes of this paper, and this subject will not be explained in detail. A rudimentary knowledge of basic language theory as taught in elementary computer science, however, will be assumed.

A parser is a software component that can read data conforming to some grammar and build an in-memory representation of the structure (or generate events based on the structures found in the language). A parser that builds an in-memory representation of the data will typically be slower than an event-based parser. And it cannot handle as large documents as the other parser type that can discard data as soon as it has recognized and generated an appropriate event. It is possible to combine the two parser types, however, so that generally the stream-based parser is used to scan quickly to an interesting part after which an in-memory representation of that part is built.

The in-memory representation of data (or events) is easier to handle programmatically than the raw data, and it makes it possible to change the document language somewhat without requiring a rewrite of the components that use the processed structures. The parser can also detect errors in the document (i.e. if the document does not conform to the expected grammar), and stop further processing because the data would appear to be corrupted. Application programmers do not need to worry about certain kinds of errors because of this parser feature.

Standard Generalized Markup Language documents belong to the category of internally marked-up structured documents. Some people have voiced their opinions that internal markup is too limiting and that SGML should be revised to offer external markup [Nel97].

3.3 Standard Generalized Markup Language

Standard Generalized Markup Language, or SGML [ISO86] for short, is an international standard for structured documents. To be more exact, SGML is a metalanguage which means that SGML is used to describe other (structural) languages.

3.3.1 A Brief History of SGML

Before going into the gritty details, let us take a journey into the history of SGML (the following is collected from [Gol90]). Even before computers existed, manuscripts were annotated with special comments to describe how the text should appear. These special comments were called `markup`. Electronic manuscripts also contained these control codes or macros. These codes could be said to be **specific coding**. **Generic coding** began in the late 1960s, the most visible change being that macros and codes got names like `heading` instead of some obscure label or directive like `format-13A3`.

The credit for this change is often given to William Tunnicliffe, who gave a speech in 1967 on the topic of separating the information content of documents from the formatting rules. At about the same time a book designer named Stanley Rice presented his idea of a "universal catalog of parametrized **editorial structure** tags". Norman Scharpf (director of the Graphic Communications Association) realized the importance of these developments and began promoting the creation of standards in this area.

Charles Goldfarb, together with Edward Mosher and Raymond Lorie, was working on an IBM research which claimed to enable text editing, formatting and information retrieval subsystems to work together and to share documents. The result of their work was Generalized Markup Language (GML). GML was based on the ideas of Tunnicliffe and Rice, but it went further, introducing formal document types and nested element structure.

In 1978 Charles Goldfarb, who had continued his research even after GML was finished, joined the Computer Languages for the Processing of Text committee under the American National Standards Institute (ANSI). Eventually he was leading the development of the SGML standard. The first working draft was published in 1980, and after several more drafts and recommendations for industry standards, SGML was finally published as an International Standards of Organization standard ISO 8897:1986.

3.3.2 Structure Is Not Layout

It has been said that all documents have structure, but let's take a memorandum for an example. A memorandum may have a title, date, author and the actual content of the memorandum. Structure can be considered to be a part of the metadata of a document. Metadata is simply information about information, i.e. what it deals with, how the information is stored, in what order certain items appear and so forth. Metadata is **not** the actual **content** of the document. The structural information in an SGML document instance is called `markup`.

Structure should **not** be confused with the **layout** of a document, although the structure of documents is usually emphasized with special layout. For example, titles in this document are printed with a larger font than the rest of the text.

3.3.3 SGML in a Nutshell

SGML is used to create vocabularies for real document languages. The vocabulary specifies what names can be used in the language. Similarly, the grammar specifies in what order the elements can appear and if they can repeat and so forth. The defined names can also have meta-information associated with them. Usually, the names are container objects that can contain other containers and plain text. The languages defined by SGML are typically infinite in the sense that one cannot write out all instances of documents conforming to a given language. The languages defined by SGML are not regular, although the content model of an element is regular. One cannot use SGML to define context-free grammars [Pre98].

SGML stores structural metadata information about classes of documents in Document Type Definitions (DTD). An actual document that conforms to a certain DTD is called an SGML document instance. In short, all real SGML documents are instances of SGML documents conforming to certain DTDs.

SGML documents have three parts in them: the **SGML declaration**, the **document type definition** and the **SGML instance** conforming to the document type definition (see Figure 4). Example 4 shows the document in text form (the SGML declaration is omitted for brevity). The three parts can be in a single file or in separate files.

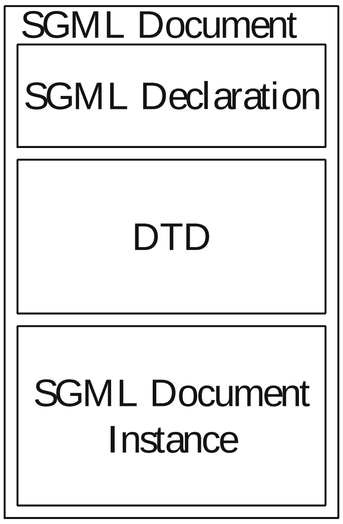


Figure 4: SGML Document Diagram.

If the declaration or DTD is to be reused in other document instances, it is of course advisable to separate them. This poses some problems, because SGML itself does not specify how the processing application can find the different parts. One widely accepted method is to refer to the Document Type Definition in the document instance with a public identifier. Public identifiers are mapped to actual file names and locations thruogh a **catalog**, typically a simple text file. Example 3 shows the contents of a sample catalog file. This ad hoc standard has been proposed by the OASIS[1] vendor consortium.

The catalog can be used to locate the SGML declaration as well. However, the SGML standard defines a **reference concrete syntax**, a declaration that is assumed if no declaration is given. The SGML declaration defines — among other things — what characters are used to distinguish the SGML markup from the actual text.

Example 3: Contents of a Sample CATALOG File.

```
PUBLIC “-//Heikki Toivonen//DTD Memo//EN” “memo.dtd”
```

Usually the reference concrete syntax is enough for most documents, although it has some rather frustrating limitations like name length limited to eight characters. Nowadays most software packages use the reference concrete syntax as a base, but extend it to enable longer names and remove some historical remnants that were required in the early days of SGML when computers where not as powerful as they are today.

The SGML declaration is not very interesting for the normal user of SGML. Instead, the DTD and document instances are more important. The ability to read and understand DTDs and document instances is sufficient for writers [Tur96].

3.3.4 DTD And Document Instance

Refer to Example 4 for clarification of the following definitions.

The reference concrete syntax specifies that declarations in a DTD start with `<!` and end with `>`. After the `<!` comes a keyword specifying what sort of thing is being declared. Usually the keyword is `ELEMENT`, `ATTLIST` or `ENTITY`. `ELEMENT` declares an element name that can be used in the SGML instance, `ATTLIST` defines attributes for an element and `ENTITY` can define different sorts of entities (like references to external images or video sequences).

In an element declaration, after the element name, it is possible to specify if the start and/or end tag of the element can be omitted. An end tag can be omitted when the parser can detect from the next element that the current element must be closed. No look ahead is needed or even allowed in the parser. The tag omit rules are specified with a minus (–) and letter `o`, `o` meaning omissible.

The second to the last part of the element declaration is the content model. The content model can

1. OASIS was formerly known as SGML-Open.

contain other element names and/or special keywords. Element names are separated with a comma (`,`) if the elements must appear one after the other, bar (`|`) if only one is allowed and ampersand (`&`) if the elements can appear in any order. It is possible to specify that an element is optional by putting a question mark (`?`) after the element's name. It is also possible to say that an element must occur one or more times with a plus (`+`), or zero or more times with an asterisk (`*`). Special keywords common in the content model parts are `#PCDATA` and `EMPTY`. `#PCDATA` means parseable character data, i.e. normal text. An `EMPTY` content model cannot be combined with any other content model element names or keywords as it means that the element does not have any content. In that case information about the element is only in its attributes. Parenthesis can be used to group parts of the content model.

The last part of the element declaration can be used to specify inclusion or exclusion exceptions. Inclusion exceptions are elements that can appear anywhere inside the element being defined, including its children. Exclusion exceptions disallow the exclusion element from anywhere in the defined element, including its children. Exclusion has higher precedence that inclusion. Typical inclusion exceptions are page break and cross-reference elements that can often appear anywhere. A typical exclusion exception would deny page break elements inside title elements.

An element can have multiple attributes. Attributes have a name, a data type and a keyword specifying if the attribute is required (`#REQUIRED`) or optional (`#IMPLIED`). An attribute can also have a default value. Common data types are `CDATA` (for normal text) and `NUMBER` (obviously a number) as well as `ID` and `IDREF` for cross-referencing. Data types can also indicate multiple entries, for example `NUMBERS` and `IDREFS`. Multiple entries are separated with spaces. If an attribute value has spaces or other separator characters in it (as specified in the SGML declaration), the value(s) must be quoted. The reference concrete syntax specifies two legal quote characters, both quote (`"`) and apostrophe (`'`).

Usually entities declared in the DTD are parameter entities, which are used to make the DTD easier to read and manage. Entities declared in a **document declaration subset** are normally used to refer to external SGML document fragments and images. The document declaration subset is enclosed within square brackets (`[` and `]`) in the beginning of the document instance. Comments appear between `<!––` and `––>`.

The SGML document instance contains the document's data. The data is mixed with the SGML markup. The reference concrete syntax specifies that start tags start with `<` and end with `>`. End tags start with `</`. Attributes are specified in the start tag between the element name and the closing `>`. To put it simply, an attribute is a name-value pair. The name and value are separated by `=`. SGML also allows only the attribute value to appear if there can be no ambiguity as to which attribute is intended.

Example 4 shows an SGML document with its DTD (the SGML declaration has been omitted for brevity).

The DTD starts with the `DOCTYPE` keyword, followed by a public identifier (the identifier could be omitted, or there could be a system identifier or both a public and a system identifier). The public identifier can be used in other documents to refer to this DTD. In the DTD there is first a definition for a notation called `GIF`, with a system identifier for the notation. The notation is later used in the definition that defines an image entity, where the image type is `GIF`.

The element definitions start with `memo` (the order of definitions is not important, by the way). It has a required start tag and an optional end tag. The element `memo` can contain an optional `title`, followed by one or more `para`s and zero or more `image`s. Element `memo` has three attributes: an optional `language`, a required `id` and `secret`, which can have two possible values with `open` being the default value. Other elements are defined similarly. The `image` element deserves special mention. It has a content model of `EMPTY`, meaning it cannot contain any other elements or text. The information for that element is in its attributes. In this case, the element has one attribute `pic`, of type `ENTITY`, and the attribute must always be specified in the document instance.

The document instance begins with the `memo` start tag. The instance shows all the required start and end tags of the elements. `para` and `image` do not have their end tags, which is legal because the DTD allows this. All required attributes are also specified. The `pic` attribute of the `image` element refers to the `tooth` entity specified in the DTD, and this should show up as an image of a tooth in an SGML browser.

Example 4: SGML Document Structure As Text.

```
<!SGML ... >

<!DOCTYPE memo PUBLIC "-//Heikki Toivonen//DTD Memo//EN" [

<!NOTATION GIF SYSTEM>
<!ENTITY tooth SYSTEM "tooth.gif" NDATA GIF >

<!ELEMENT memo - o (title? , para+, image*) >
<!ATTLIST memo
language CDATA #IMPLIED
id ID #REQUIRED
secret (open | internal) "open" >

<!ELEMENT title - o (#PCDATA)>

<!ELEMENT para - o (#PCDATA)>

<!ELEMEMT image - o EMPTY>
<!ATTLIST image
pic ENTITY #REQUIRED>

]>

<memo id="unique-id-1" language="English">
<title>Remember Dentist!</title>
<para>Dentist tomorrow at one o’clock.
```

```
<image pic='tooth'>
</memo>
```

3.3.5 External Entities — A Simple Way to Reuse And Manage Text Fragments

External entities are such a critical part of this thesis that it is important to get the basics right. An external entity declaration contains the entity name followed by a public identifier or a system identifier or both. Additionally, a notation type can be specified. If there is no system identifier, the processing application must somehow get the system identifier and acquire the contents of the entity. The OASIS catalog is a good way to map public identifiers to system identifiers. It should be noted that a system identifier is exactly what the name says. It is not necessarily a file name. It could be a database query or just about anything. As long as the processing system knows what to do with it, it can use any means or description desired to cause the contents of the entity to be made available to the SGML processor.

An entity is used in a document by simply inserting the entity's name, surrounded by an ampersand (&) and semi-colon (`;`), where it is wanted. Similarly, special characters that are not present in the character set used can be inserted into the document. So, for example, the letter `ä` is usually represented with an “umlaut” entity called `auml` as `&auml;`.

It is possible to define a default entity whose contents will be used for entities that are not defined. The entity text could say for example `@@Undefined entity@@` to indicate that it should be defined at some point. This is sometimes used to make documents containing undefined entities valid.

External entities allow reuse of parts of documents in other documents even if the files are saved in a normal file system. For example, a manual consisting of several chapters could be constructed so that each chapter would reside in its own file. The whole manual would need to be assembled from these files. How this might look like in SGML is shown in Example 5. Now, if there were several manuals and the first chapter was an introduction that was identical in all manuals it would be a simple matter to reuse the first chapter. Of course this saves space, but the more important bonus is that by changing the first chapter all documents using it would be immediately updated. Automatic update may not be wanted in every case but it is impossible to prevent this without additional tools if reuse is what is wanted. Unfortunately reality sucks, and not all SGML tools support this external entity approach very well even though it is quite simple.

Example 5: External Entities.

File `manual.sgm`:

```
<!DOCTYPE manual PUBLIC “-//Heikki Toivonen//DTD Manual//EN“ [
<!ENTITY chap1 SYSTEM “chap1.inc“>
<!ENTITY chap2 SYSTEM “chap2.inc“>
]>
```

```
<manual>&chap1;&chap2;</manual>
```

File `chap1.inc`:

```
<chapter><title>Introduction...</chapter>
```

File `chap2.inc`:

```
<chapter><title>The Life of Brian...</chapter>
```

The problem with the external chapters in Example 5 is that they themselves are not valid SGML documents. They need to be included in a manual that has the `DOCTYPE` header information. There is a rarely used SGML feature called `SUBDOC` that would make it possible to create the chapters as standalone documents and it would still be possible to include them in the manual file. Very few tools support it, though.

3.3.6 Users of SGML

The value of SGML was first fully realized in large corporations and organizations, like the U.S. Department of Defense. Other large companies using SGML now include Nokia, Novell, Microsoft and Hewlett-Packard.

Although the original users were mainly involved with the military, the commercial wing has caught up. The greatest interest in SGML seems to be in areas like telecommunications, aerospace, manufacturing, publishing and pharmaceuticals. This does not mean that SGML is not suited for other users. It so happens that the aforementioned areas share the characteristic that they produce huge amounts of documentation that has to be maintained and manipulated effectively. However, the amount of documentation is not the only indication that structured documents might be a good solution to better manage the information. Frequent updates, multiple authors, long term storage and stringent validation requirements are other indicators, to name a few. The Extensible Markup Language (discussed below) is rapidly bringing smaller players into the picture as well.

SGML should generally not be used where the information is used only once, it easy to reproduce, it is not important or, in short, the opposite of what was mentioned before. Typical examples of documents that would not benefit from SGML include notes and informal letters.

It may be interesting to note that HyperText Markup Language (HTML) is SGML. HTML is simply an SGML DTD. However, HTML follows rather loosely the philosophy underpining SGML. For example, HTML has page formatting elements, which are completely against the principles of SGML.

Regardless of HTMLs deviance from SGML principles, it has done a great job of advertising the benefits of structured documents, along with World Wide Web (WWW), of course. The Web itself is becoming more “mature” in a structural sense. The appearance of Extensible Markup Language enables web authors to mark up their documents with descriptive markup instead of the restricted HTML tag set. This will eventually bring smart search engines to the web that can use the document structure to achieve better signal to noise ratio [Mya98].

3.4 Extensible Markup Language

The Extensible Markup Language (XML) [W3C98] is the little brother of SGML. The credit for the birth of XML is often given to two men in particular: Jon Bosak and Tim Bray. Their hard work to bring SGML to the World Wide Web resulted in XML. It has succeeded in bringing the true power of SGML to the masses. It is easier to parse and use than SGML, which makes it easier to write XML applications, not to say faster and cheaper. All valid XML documents are valid SGML documents, but not vice versa.

XML did away with some of the historical baggage SGML was carrying around to make it easier to process. For example XML does not allow exceptions (see Section 3.3.3) nor is it possible to omit tags in XML. Because tags cannot be omitted the characters `-` and `O` just after the element name in element definition are no longer used in an XML DTD.

The only valid character set is Unicode [ISO93], while in SGML the character set can be specified in the SGML declaration. Because the Unicode code base could potentially include practically all characters there are or ever will be, no character entities are used — meaning `ä` is encoded as `ä` instead of `&auml;`. Numerical entities can still be used (for example for characters that cannot be typed). Numerical entities are of the form `&#<number>;` where `<number>` is some character code in Unicode. Also, all names are case sensitive (SGML allows one to specify if case is relevant). There are plenty of these small changes, but generally one does not need to worry about them unless there is a constant need to do transformations between SGML and XML.

XML introduced the concept of **well-formedness**. All XML documents must be well-formed. A well-formed XML document has all its open and close tags, elements nest properly (i.e. a situation like `<a><b></a></b>` is prohibited) and it does not need to have a DTD. A **valid** document is naturally well-formed, but it must also contain a DTD and it must conform to the DTD. Parsers that check only well-formedness are pretty easy to write, and they can work a lot faster and with less resources than validating parsers. See Example 6 and Example 7 for samples of well-formed and valid XML instances, respectively.

Example 6: A Well-formed XML Document.

```
<?xml version='1.0'?>
<memo>
<title>This is a title</title>
<para>I have to remember this</para>
</memo>
```

Example 7: A Valid XML Document.

```
<?xml version='1.0'?>
<!DOCTYPE memo [
<!ELEMENT memo (title,para+)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT para (#PCDATA)>
]>
<memo>
<title>This is a title</title>
<para>I have to remember this</para>
</memo>
```

It is simple to delinate where SGML is and is not an appropriate technology option but it is less obvious with XML. Because well-formed documents are easy to process and do not require strict adherence to any predefined DTD it could be argued that XML could be used everywhere.

XML itself is really a clear-cut standard as standards go (strictly speaking it is not a standard per se, but a World Wide Web Consortium Recommendation which is effectively an Internet standard). But XML does not exist in a vacuum, and there are many related Recommendations in the works or already finished that are usually needed when working with it. The Namespaces in XML [W3C99a] Recommendation is a sort of an effort to simplify the architectural forms concept from the HyTime standard (see Section 3.5). The XML Linking and Pointer activity tries to create a simple but powerful linking and addressing model for XML (again, HyTime is the ultimate system but it is too complex). The Extensible Style Language (XSL) [W3C99b] is an effort to create a powerful page description and transformation language (based on XML) that can be used to transform XML documents to screen and paper representations, for example. And there are still others.

Having mentioned XSL, it must be said that there are existing standards and Recommendations that do what XSL is trying to achieve. DSSSL (see Section 3.5) obviously is up to the task, but it is too complex. Cascading Style Sheets (CSS) [W3C96] can do a lot, but CSS has very limited transformation capabilities. For example, it is not possible to create a table of contents with a CSS stylesheet. Interactive documents will always need scripting. The scripting language could be for example JavaScript, or the standardized version ECMAScript [ISO98b]. There has been a heated debate on the need for XSL[1].

3.5 HyTime

HyTime [ISO97] is the abbreviation for Hypermedia/Time-based Structuring Language. In an effort to develop a standard representation for music with SGML it was noted that SGML alone was not up to the task. Also many things that deal with music can be used with other time-dependant systems, such as multimedia. Hypermedia poses some difficult challenges to the software that must operate on it [Kim98], like how to create non-character content, how to schedule and render real-time content, how to locate specific objects and parts of objects within the data. So in order to cater for a variety of different needs, HyTime was first developed. The first version of HyTime appeared in 1994. The current version was published in 1997.

1. See the discussion forum at XML.COM, `http://www.xml.com/`.

3.5.1 Hypermedia Concepts and Dimensions

Some of the basic concepts of hypermedia are not new. There have been cross-references in books even before print was invented. Vannevar Bush wrote the first paper describing an automated hypertext system [Bus45]. The term hypermedia is rather new, however, as it has only emerged in the recent decades. It is worth mentioning that the term **hypertext** coined by Ted Nelson in the 1960s and discussed in [Nel82] really means **hypermedia** in its original sense.

It is possible to draw a dimension diagram for hypermedia (see Figure 5, redrawn from [Kim98]). HyTime is most applicable for the top two quadrants of the diagram ([New91] and [Kim98]). Still, it has been claimed that HyTime's scope of applicability is comprehensive enough to embrace all possible text processing applications ([New91] and [DeR94]). Therefore all documents can be represented with HyTime[1]. This sounds like a ridiculous statement, but with a bit closer look it is not that difficult to image that this could be true.

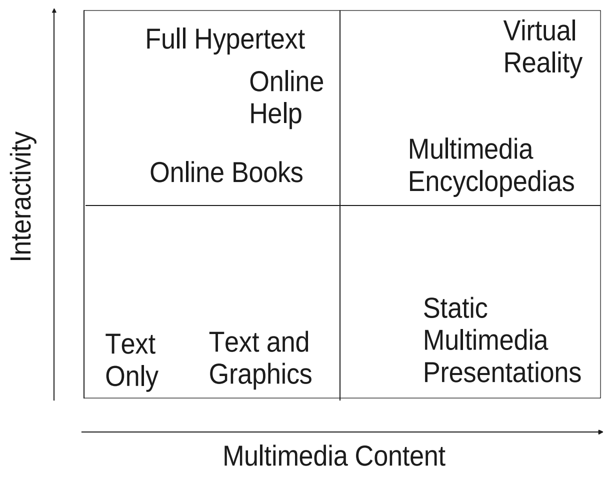


Figure 5: Dimensions of Hypermedia.

3.5.2 HyTime Hyperdocuments

Every HyTime **hyperdocument** has a so called **hub-document**, which is an SGML document with some additional HyTime constructs. The additional HyTime constructs can refer to external entities, like video sequences. If there is a video sequence we would like to represent with HyTime we do not actually need to convert it to SGML, but instead there must be a process that can understand the video format, and provide a **grove** representation of it to the HyTime engine (all HyTime addressing and linking happens in the grove).

A grove is a formal construct from the HyTime standard. A grove is roughly a parse tree and some additional information. If we would like to view the video with a program that has an embedded

1. "Resistance is futile, you will be assimilated."

HyTime engine, we would open the hub-document, which would let the HyTime engine launch the actual viewer for the video. The grove constructor would be needed if we would need to point to some frame in the video sequence, for example.

HyTime is a large standard, but, fortunately, highly modular. This means that one can implement very small subsets of it. One such module from the standard is the HyTime hyperlink module. The HyTime hyperlinking mechanism is one of the most widely used features of HyTime as it is relatively easy to implement on top of existing SGML systems and it is really useful. With SGML one can only link from an element to other uniquely identified element(s) in the same document instance. HyTime's links do not have that kind of restrictions.

3.5.3 HyTime Markup

Arguably the most simple HyTime link construct is the **contextual link** (clink). It is a deceptively simple yet powerful concept. `clink` can link to both internal locations and to external documents and locations in them. The **link initiating anchor** is the link itself, i.e. it happens in context (hence the name contextual link). Typically the link begins as an SGML `ID`/`IDREF`, although other forms are possible. The target `ID` is often just a pseudo-target in the same document to satisfy standard SGML parsers. HyTime engines know to look more closely at the pseudo-target to see if it is a part of a location ladder or path that eventually points out the real target.

Example 8 show how `clink`s could appear in SGML markup. Most of the DTD has been omitted for brevity. The ellipsis (`...`) marks deleted sections.

The `xref` element is a `clink`. HyTime does not require the use of specific element names. The specific HyTime constructs are indicated by attributes. By default, the attribute name is `HyTime`. The `xref` element has a required linkend attribute of type `IDREF` (meaning it must point to a unique identifier in this document). The `HyTime` attribute has a fixed default value. A fixed value means that there is no other legal values for this attribute other than the one specified in the attribute definition.

The `nameloc` and `nmlist` elements are defined so that they conform to similarly named constructs in the HyTime standard. Their purpose is to enable linking to other named locations in this document or other documents. Example 8 shows how to link to other documents. The `nameloc` element provides a target for the `xref` `IDREF` linkend so that SGML parsers will find this document valid. The real target, however, is something else as specified by the HyTime standard.

Example 8: HyTime clinks.

```
<!DOCTYPE hydoc PUBLIC “-//Heikki Toivonen//DTD My HyTime Doc//EN“ [
...
<!ATTLIST xref
linkend IDREF #REQUIRED
HyTime NAME #FIXED “clink“>
...
<!ATTLIST nameloc
id ID #REQUIRED
HyTime NAME #FIXED “nameloc“>
...
<!ATTLIST nmlist
docorsub ENTITY #IMPLIED
nametype (element|element) “element“
HyTime NAME #FIXED “nmlist“>
<!ENTITY otherdoc SYSTEM “otherdoc.sgm“ CDATA SGML>
]>
<hydoc id=”id-1”>
<title>My Hydoc</title>
<para>Link to ID <xref linkend=”id-1”>“id-1”</xref>.</para>
<para>Link to <xref linkend=”loc-1”>“otherdoc.sgm“</xref>.<para>
<para>Link to <xref linkend=”loc2”>“id-251” in “otherdoc.sgm”
</xref>.</para>
<hylinks>
<nameloc id=”loc-1”>
<nmlist nametype=”entity”>otherdoc</nmlist>
</nameloc>
<nameloc id=”loc-2”>
<nmlist docorsub=”otherdoc” nametype=”element”>id-251</nmlist>
</nameloc>
</hylinks>
</hydoc>
```

3.5.4 Architectural Forms

Another frequently used construct from the HyTime standard is the **architectural form**, or **meta-DTD**. Meta-DTDs bring object oriented thinking to document management, i.e. document types can inherit certain properties from their ancestor DTDs. In this regard they act a bit like base classes or supertypes in object oriented programming languages [Kim97]. In fact, Example 8 uses the HyTime meta-DTD, from which the `clink`, `nameloc` and `nmlist` forms are instantiated.

The architectural forms facility is a great help in document management and interchange. With vanilla SGML an author is stuck with a given DTD and can not enhance it for his own special needs (or if he does enhance the DTD, his documents may become unusable to other users of the original DTD). Architectural forms allow the authors to enhance the document structure to better suit their needs while still enabling document interchange. The designers of the meta-DTDs can also specify certain constraints on the derived DTDs.

Document Style Semantics and Specification Language (DSSSL) [ISO96] is very closely related to HyTime. They share the same grove model of the document. DSSSL is generally recognized as a page layout language, but because it really is a programming language, it enables very sophisticated modification of SGML/HyTime documents. Its query language can be used in the query location address form of HyTime.

HyTime has gained the reputation of being a difficult and expensive technology. While it is true that the standard in its entirety is intimidating many bits and pieces are relatively easy to understand and implement, and thus use. There are not too many resources for learning HyTime, but one of the best is [Kim98].

3.6 Related Standards

SGML is not the only standard for structured documents. Open Document Architecture (ODA) [ISO89] addresses the same problems as SGML. ODA is somewhat more complicated, which is probably why it has not been as widely accepted as SGML. The main difference between ODA and SGML is that ODA can also be used to store the layout information of documents. Originally ODA was called Office Document Architecture, but this was later changed to Open Document Architecture.

Virtual Reality Markup Language (VRML) [ISO98a] has been developed to describe virtual reality spaces and objects. There are also structural standards for music, chemical patterns, product information and exchange and so on. The simplicity of XML attracts many of these other standards, and there are investigations going on to convert them to XML or at least provide a mapping to XML structures.

Most documents contain illustrations. Effectively managing images is at least as important as managing text. Therefore it is no wonder that there are several international and de facto industry standards available for images. It has often being said that the Computer Graphics Metafile (CGM) [ISO92b] standard is the equivalent of SGML for 2-dimensional images. The drawings in this document are mostly in CGM format.

Chapter 4

Databases

We can still remember the golden days before Heisenberg, who showed humans the walls enclosing our predestined arguments. The lives within me find this amusing. Knowledge, you see, has no uses without purpose, but purpose is what builds enclosing walls.

Frank Herbert, *Children of Dune*

Any collection of data can be considered to form some sort of a data repository or database. Nowadays the word database is almost exclusively reserved for an electronic collection of data that is managed by a very special computer program, the database management system (DBMS). The technical sources for this chapter were mostly [Sun81], [Mic92], [Loo98], [Whi99].

4.1 Common Database Properties

Databases have been around for a long time, even computerized databases have been in existence for decades now. They serve as organized information repositories.

Databases are scalable. They are usable as small, single person databases. A listing of one's personal video collection could be one example of such a micro-database. Largest existing databases easily exceed a petabyte in size. This does not mean that any database software can be used to manage a database of arbitrary size.

The most important and useful property of databases is the ability to perform queries on the data efficiently, filtering out unwanted information. Queries typically execute very fast. Of course these properties are a must in a large database, otherwise it would be unusable. Databases can also save space by eliminating redundant information, and there are techniques to help validate the data in databases.

Typically databases are used in situations where the database structure does not change, but the information in the database can change rapidly. A common example is a bank's customer database as the information describing a customer, or the schema, is not likely to change over many years. Still a person may make several withdrawals and deposits in a single day, all of which must be reflected in the database.

The current mainstream database technology is divided into relational and object-oriented database models (discounting models that are obsolete or nearly so)[1]. There are some databases on the market that are referred to as **hybrid** databases. That means they are not strictly relational nor object-oriented. A typical hybrid database evolves from a pure relational database when a variable length text field is introduced (possibly with grammar check, for example by an SGML parser) [Elo95].

Before going futher it should be noted that the term database used in this paper actually means the database management system (DBMS) and the data as one entity. Most references clearly separate these concepts and only talk about a database when the data is meant.

4.2 Relational Database Model

The relational database model is based on mathematics. Besides making it elegant by design, it has resulted in practical benefits. For example, certain things can be proven which helps in validation and optimization. This section will not even try to define the concepts as they really should be defined from a mathematician's point of view, but rather from the view of the every day user and software designer who only needs to use databases, not design them. Moreover, the emphasis is on data retrieval, although databases handle simultaneous updates to data as well. On a file system simultaneous updates easily corrupt the data.

The end of the section lists some references that will explain the relational model more deeply and with mathematics.

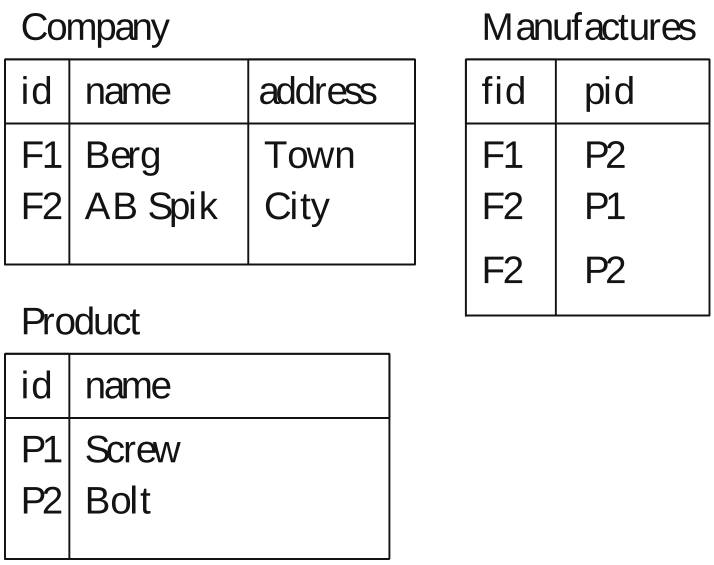
It should be noted that most "relational databases" on the market are in fact not fully relational. Dr. Edgar Codd's ("the Father of the Relational Database") work in the 1970's and 1980's resulted in 13 rules that a relational database should fulfill. When the relational concept gained accep-

1. Discounting also the World Wide Web, even though it is a sort of a networked database. While WWW fills the basics of a database definition, it lacks for example an effective querying mechanism.

tance, the database vendors quickly implemented some properties from the relational theory and happily sold their products as relational databases. However, this does not imply that the databases are not well suited for their job. And in fact, the solutions presented in this paper demand very little from the databases used.

4.2.1 Basic Building Blocks

In the relational model, data concerning a given entity is collected in a table. Take, for example, a (simplified) database describing companies, products and information about the relationships between the companies and the products they make. The Figure 6 shows a relational database with three tables: `Company`, `Product` and `Manufactures`. Tables consist of columns. For example the `Company` table consists of `id`, `name` and `address` columns. Rows in a table contain the actual data in the table, while the tables and columns (or their names) themselves are basically metadata.



Company

id	name	address
F1	Berg	Town
F2	AB Spik	City

Manufactures

fid	pid
F1	P2
F2	P1
F2	P2

Product

id	name
P1	Screw
P2	Bolt

Figure 6: Relational Database Model.

In typical database implementations some of the columns in a table form a so called **primary key**, which must be unique for each row in a table (the relational database theory does not require this, but this makes it easier to optimize the database performance). In the example the `id` column of the `Company` table is the primary key. Relationships between tables can be described as links between primary and **foreign key** columns. A foreign key in a table “points to” a primary key of another table. The relationships between companies and products in the example are described in the table `Manufactures`. So to see what products can be found from a company one must first follow the link from the `Company` table to `Manufactures` and from there to `Product`. In this example the company `Berg` manufactures only `Bolts`.

Relationships can be **one-to-one**, **one-to-many** or **many-to-many**. A one-to-one relation means that for each row in the left-hand side table of the relationship there exists a maximum of one row in the right-hand side table. A one-to-many relationship is simply a relation where there can be multiple "hits" in the right-hand table. Many-to-many relationships are created with helper tables. For example, the `Manufactures` table in Figure 6 is a helper table. There is a one-to-many relationship from table `Company` to it, and there is a one-to-many relationship from table `Product` to it. One-to-one relationships are rare, but the other two forms of relationships exist in most databases.

In diagrammatic models of databases relationships between tables are sometimes indicated by lines drawn between them (entity relationship or ER diagrams). The type of relationship is usually indicated with numbers and other symbols. For example, one-to-many relationship could be indicated by a "`1`" on the left-hand side and by "`n`", "∞" or "`1..*`" on the right-hand side. See Figure 18 in Section 6.3 for an example.

Relational database can also be thought to consist of different levels, or components. On the lowest level there is the physical storage level. The physical representation of databases is a science in itself. Usually binary formats are used, but above the binary level the database can be represented as sorted or unsorted indices, various trees and other constructs. The logical level of a relational database is the table level, or in a broader sense the query level. The upper level is the reports, or user-interface level. Each level uses the levels beneath it to carry out its purpose. For example, a report has some knows how information should be shown and it can contain complex calculations on the data. Reports use queries to fetch the data. Eventually queries access the physical data on a storage device. Finally there can be the management system for all of the "components". A typical off-the-shelf database like Microsoft Access comes with all of these.

Relational database systems have methods that try to ensure that the data in the database will always be correct. For example it can be made impossible to remove a company from the database without removing the products it manufactures (and are not manufactured by any other company). This kind of control can be achieved by checking the primary and foreign keys, and making sure that if a key in a table is deleted it is not referenced anywhere else.

4.2.2 Normalization

It is possible to design a database badly. Bad design means that the database contains redundant information, there are things that cannot be queried from it and so on. The process that tries to avoid these kinds of problems is called **normalization**. Normalization is done in steps. We say that a table is in **first normal form** (1NF) if all of the data values are atomic values. Achieving 1NF is mainly common sense, and modern RDBMS make it difficult to create tables that are not in 1NF [Whi99].

A table is in **second normal form** (2NF) if it is already in 1NF and all non-key fields (columns) are fully functionally dependent on the primary key. As an example of functional dependency, consider the volume of a box, which can be calculated from its dimensions. If the dimensions are stored in the table, then the volume should not be stored because it is functionally dependent on the dimensions. Stepping from a lower normal form to a higher normal form is carried out by splitting the table.

The **third normal form** (3NF) is achieved when the table is already in the 2NF and all non-key fields are non-transiently dependent on the primary key. This simply means that any non-key field should depend solely on the primary key. The `Company` table in Figure 6 would be an example of this as long as the `addressID` column is always dependent on the `address` column. There are at least three other levels of normal form, but generally a database in the 3NF is considered good enough [Whi99]. The database in Figure 6 is at least in the 3NF because all the conditions are fulfilled.

4.2.3 Queries And Beyond

To get data out of multiple tables or restrict what comes out of a single table, a query is needed. Actually SQL, the Structured Query Language [ISO92a], can be used to create and alter the database structure as well as inserting and modifying information. Consider the example in Figure 6. One might want to know what products the company `AB Spik` has in its catalog. This could be carried out with the SQL commands presented in Example 9.

Example 9: SQL `SELECT` Statement.

```
SELECT
Product.name
FROM
Company, Manufactures, Product
WHERE
Company.name = “AB Spik”
AND Company.id = Manufactures.fid
AND Manufactures.pid = Product.id;
```

In English, the above query says:

> This query operates on tables `Company`, `Manufactures` and `Product`. Find the `id` of the company whose name is `AB Spik`. Next, use the found value to locate the rows in the `Manufactures` table that have that value in the `fid` column. Finally, for all the rows found in `Manufactures`, find the rows in the `Product` table that match the `id` column values with the `pid` column values in the `Manufactures` table, and show their names.

Saying it aloud is quite a mouthful, but it really is quite simple. Initially selecting the three tables produces a result set that has all the possible combinations available in the database. The `WHERE` rules further restrict the result set, because the `AND` keyword means that all of the rules must be true simultaneously. From the result set a single column is selected for output.

A nice tutorial on SQL is [Hof99]. Recommended reading about relational databases in general are [Whi99] and [Yar99], the latter because it has examples using a freeware relational database. However, the accuracy of information is not always as good as it should be. Finally, [Loi99] is a book for the database freak. It goes into the mathematics of relational databases.

4.3 Other Database Models

Relational databases are not the only databases on the market. Object-oriented databases are steadily gaining ground. Evolution has nearly discarded some database technologies, while research for new and better technologies goes on.

4.3.1 Object Oriented Databases

Objects in an object-oriented (OO) database may have data attributes and other objects. In fact, one definition of an object oriented database is a "collection of persistent objects" (that is, objects that live between invocations of a program) [Eck95]. This could be depicted as shown in Figure 7. The outer ellipse describes the database. The database holds objects like a circle and a box. There are some objects that in turn consist of other objects, like the ellipse on the left that consists of two sub-objects. The figure is just an abstract visualization of an OO database and does not imply that data is stored as images.

Contrary to popular beliefs, there are excellent object-oriented databases available. OO databases are being used, and they can offer easier handling of data and better performance, to name just a couple of benefits over relational databases [Wak99]. OO databases are naturally suitable for storing structured documents ([Paq92], [Böh94] and [Bal97]), which is also proven by the available SGML/XML databases using OO solutions (for example, Astoria[1] and POET Content management Suite[2]).

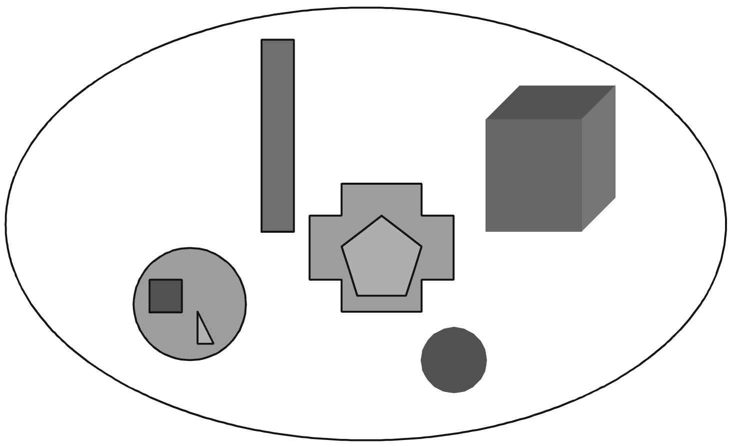


Figure 7: Object Oriented Database.

OO databases also have the benefit that developers are not faced with "impedance mismatch". Impedance mismatch is a term used to describe the problems inherent in using SQL language

1. Astoria is a product of Chrystal Software, a division of Xerox.
2. POET Content Management Suite is a product of POET Software.

from within an OO language like C++ and binding SQL's structural constructs to classes and objects. Using objects stored in an OO database within an OO programming language can work with exactly the same rules and syntax as working with any other object.

One serious handicap with OO databases is that they are not based on such a rich mathematical foundation as the relational model. This makes it more difficult to optimize them and present standardized solutions. It remains to be seen if pure OO databases will enter the mainstream. One way to get there might be to build an OO database on top of a relational database. This has been discussed in [Loo98] and [Loi99].

4.3.2 More Exotic Database Models

Relational and object oriented databases are certainly not the only database technologies out there. Other technologies have been in use before them, and new technologies are being invented.

Hybrid databases were mentioned earlier. A hybrid database is a hybrid between relational and object oriented databases. To make it clear that the techniques involved are relational and object oriented, these databases are often referred to as object-relational databases. It has been proposed that SQL be further developed to be more compatible with object-relational databases. It has been realized that pure relational databases are lacking in several important areas, for example in handling structural data [Rei98].

Hierarchical database technology (see Figure 8 for an illustration) has lost the technology war to the newer technologies. On the other hand, recent discussion in some XML forums indicates that it might again prove to be a viable solution in some specific cases. The network database model (see Figure 9) is almost unheard of except with old mainframe computers. References [Sun81] and [Loi99] discuss hierarchical and network models in more detail.

A brief visualization of the differences between relational, hierarchical and network models can be seen from Figure 6, Figure 8 and Figure 9 (modified samples from [Sun81]). The figures themselves do not explain the different database models, they just show that different ways to represent the same information exist. The sample database has information about companies and products, and describes what company manufactures a given product.

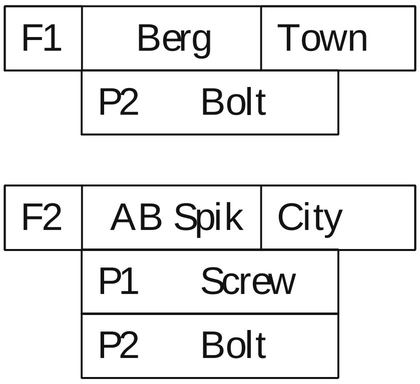


Figure 8: Hierarchical Database Model.

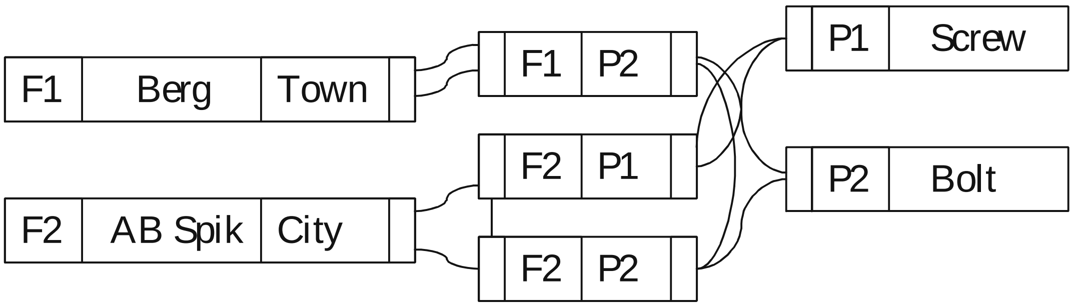


Figure 9: Network Database Model.

Chapter 5

Managing Documents with Databases

You will learn the integrated communication methods as you complete the next step in your mentat education. This is a gestalten function which will overlay data paths in your awareness, resolving complexities and masses of input from the mentat index-catalogue techniques which you have already mastered. Your initial problem will be the breaking tension arising from the divergent assembly of minutiae/data on specialized subjects. Be warned. Without mentat overlay integration, you can be immersed in the Babel Problem, which is the label we give to the omnipresent dangers of achieving wrong combinations from accurate information.

Frank Herbert, *Children of Dune*

There are several ways to manage documents with databases. The most straight-forward way is to store the documents themselves into databases. That can be problematic, though, if the documents are large and the system can not split the documents into smaller parts. The other easy approach is to save documents normally, but have databases manage references to those documents. This thesis is mostly concerned with the latter method. The primary sources for this chapter are [Tra95], [Pel97a], [Ryt97] and [Som96].

5.1 Differences And Similarities Between Databases And Structured Documents

Documents contain information. It is important to be able to effectively manage that information. Databases were created to effectively manage small information blocks, like numbers, but they can also be used to manage whole documents or parts of documents. Depending on the database technology, the actual way of managing data differs, but the expected functionality is pretty much the same. It must be easy to find the information, restrict access to it, enable information reuse, keep track of changes and so on.

5.1.1 Storing And Retrieving Complete SGML Documents — A Challenge to Databases

It is possible to break an SGML document into relations and tables and therefore it is possible to "insert" SGML documents into relational databases. However, the result is probably not what one usually calls a document. With special programs it is of course possible to view the relational representation in a standard SGML document way. This approach is sometimes used with very large SGML documents that have to be managed effectively.

It is relatively easy to understand how SGML documents can be saved in OO databases. Each element in an SGML document is an object. Container objects are formed of several smaller objects. It is no wonder then that there are several OO databases specialized in SGML, for example Astoria and POET.

Databases are very different from SGML documents. Whereas data in an SGML file resides in sequential order as ASCII characters, databases use several different methods for storing the data (see Section 4.2.1).

Although it would be possible to describe a database's structure with a DTD and to display the data as a document instance, this is not generally what is wanted. That kind of report can be easily obtained from the database itself, even though it is not in SGML.

With relational databases, showing the contents of the database as SGML tables, possibly with links, would require a considerable amount of work from the user in order to find the answers to questions he or she might have in mind. On the other hand, queries can be constructed so that they answer specific questions (see Section 4.2 and Example 9). Depending on what the retrieved data is, it can be given a logical structure with a DTD.

Also, because databases are often huge in size, building a static SGML instance of it would take a lot of time. Needless to say, complete databases are generally too big to fit into RAM anyway. The way to handle this is to build the instance as a user is browsing it, thus possibly eliminating many needless queries. Another problem is that the data in a database may be changing in a rapid succession, while an SGML file normally is supposed to stay valid for some time, even decades. So to really make an SGML instance of a database one should ideally fetch only the information the user wants to see and update the document continuously to reflect the changes in the database.

5.1.2 Extracting Parts of Documents from Databases

There are two basic syntactic approaches to indicate in an SGML document instance that certain parts of the document should be retrieved from a database. First, an entity's system identifier (see Section 3.3.4) can be a database query instead of a file name. The entity manager must then be customized to retrieve the entity's content from a database. For example, the system identifier of the `chap1` entity in Example 5 could be the SQL statement in Example 9. The resulting generated chapter could be similar to Example 10.

Another option is to attach a query to an element. The query should be executed in order to get the element's content. Database queries can be stored in attributes, or queries can exist elsewhere and they can be referred to by HyTime links. The application processing an element should then use the query to retrieve the element's content. This approach is explored more in the DTD presented in Appendix B.

The second method looks more appealing, especially with the Second Edition of HyTime. The simple approach here is to have an attribute that contains the query to execute. This can be expanded to give some meaning to the attribute, i.e., tell the processing application what the attribute value is and also make it clear that the application should process the attribute value to get the element's content. This can all be encoded in a standard way. This approach can be made even more robust if the queries themselves are part of the document. Then it would be possible to point a link to the query wherever it is needed. This way the queries can be reused and even managed effectively in a central repository.

Another way to classify the how the database structure can be mapped to document structure is through the concepts of template-driven and model-driven mappings. In a **template-driven** mapping the query is embedded somewhere in the document. Once executed, it will replace some part with a template element structure where certain variables are replaced by database values. In the **model-driven** mapping there is some fixed document structure that is mapped to the database structures. For example, there could be a `table` element that must be mapped to a table in a relational database. The template-driven and model-driven mappings are explained in more detail in [Bou99].

As it was pointed out earlier, blind mapping of the tables and columns of a relational database to a plain table/column DTD would not result in a meaningful document. A container element should be mapped to a query, and the query output columns to some display elements in the SGML DTD. But what to do when a query returns multiple rows, as is usually the case?

A generic way would be to define a row model so that each row in a query's result set would generate a small piece of SGML tagged text, each row generating the same SGML structure but different element content. Let us take, for exampl,e a query that returns results in two columns, say

Author and Book Name. The result set contains two rows where the first row would have values Frank Herbert and Dune while the second row would have values Piers Anthony and On a Pale Horse. It might be desirable to output the result in SGML as in Example 10.

Example 10: Markup Generated from Database.

```
<book>
<author column="Author">Frank Herbert</author>
<title column="Book Name">Dune</title>
</book>
<book>
<author column="Author">Piers Anthony</author>
<title column="Book Name">On a Pale Horse</title>
</book>
```

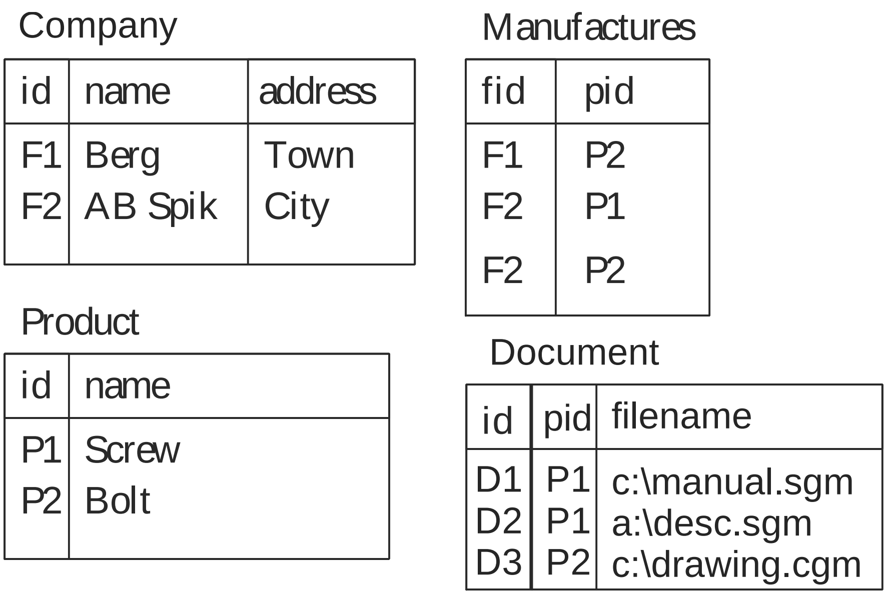
5.2 General Purpose Databases

A general purpose database means here a database not designed specifically to store and manage documents. There is a wide selection of these off-the-shelf databases available.

General purpose databases cannot really do anything smart with a large document. They must simply save it as a single large object — if they can! The text part of a single document can exceed 50 MB in some aircraft manuals. Not all databases can handle objects this big.

Another approach, if the database is not required or cannot save the whole document, is to save the document on a normal file system and only store a reference to it in the database. This is easy from the view of the database, but it is not foolproof. Someone could go and change the entry in the database without moving the file in the file system or vice versa. An additional tool to manage the reference in the database and the physical location of the file on the file system would be a good idea. If the physical location of the file was hidden or inaccessible to users without the additional tool the system could be made quite safe.

If an organization is already using a relational database to manage product data (but not product documents), it may be really simple to modify the database to make it usable as a document management system. For example, let us look at Figure 6. If there is documentation for each product, and we would like to add information about the documents into the database, we would only need to add one new table, called, in the example, Document. Let us assume that we only need to know where the document is located. In that case we would need a filename column in the table. Figure 10 shows how the database looks after this addition. Screw now has two documents associated with it, and Bolt has a drawing. This “reference approach” is used in the system implemented at Wärtsilä NSD, described in the next chapter.


Company

id	name	address
F1	Berg	Town
F2	AB Spik	City

Manufactures

fid	pid
F1	P2
F2	P1
F2	P2

Product

id	name
P1	Screw
P2	Bolt

Document

id	pid	filename
D1	P1	c:\manual.sgm
D2	P1	a:\desc.sgm
D3	P2	c:\drawing.cgm

Figure 10: `Document` Table in Relational Database.

5.3 Specialized Databases

Specialized database in this context means a database that is designed and implemented to store and manage documents. They can be further divided into systems that either are or are not designed specifically to manage structured documents.

Normal document management systems do not know that documents could contain internal structure. They just store them as blobs of data. It may be possible to specify that certain documents belong together in a certain order, along with metadata such as who created them and when, and when were they last accessed.

Version management systems commonly used in the software industry are also a bit like document management systems. They are in some sense aware of the contents of the files thrown at them, because programmers like to see the differences between different versions of files. This is often taken a bit further to optimize space, because the system need only store the original version of a file and the changes. This primarily makes sense with text files. Of course, products sold as document management systems can do things like this as well.

The most interesting document databases are the ones that understand that documents can have structure. An SGML document can be understood by its DTD and split into small objects stored efficiently in the database. The document can be reassembled back into textual representation when it is checked out.

A structured document database need not have any limitations as to how fine-grained the logical objects it handles can be. The pure SGML way of achieving document reuse is by using entities, but that presents some problems, as was discussed earlier (see Section 3.3.3). With a good database a user can lock a single paragraph from a monstrously large manual, check it out, edit it, and check it in without stopping other authors working on the same document.

Documents can be constructed from many different objects in the database, sharing some objects. As with entities, updating a shared object will immediately update all the documents that use the shared object (see Figure 11), but the system can warn the user that this is about to happen and ask if this is really wanted. If the user does not wish to update the referring documents he can instruct the database to make a copy of the original content. Documents may either refer to the original shared object or the new, modified shared object.

Of course, structured document databases can also save metadata about documents such as who created what and when. Beyond this, they are capable of taking this down to the smallest possible logical element in the document as well. Some specialized databases also include other components like workflow management systems which can automatically move a job from one person to the next as soon as work phase is completed[1]. All this makes special purpose, structure-aware databases superior to other solutions. The price tag may also far exceed other, inferior techniques. The cost balance can therefore make less comprehensive solutions still attractive.

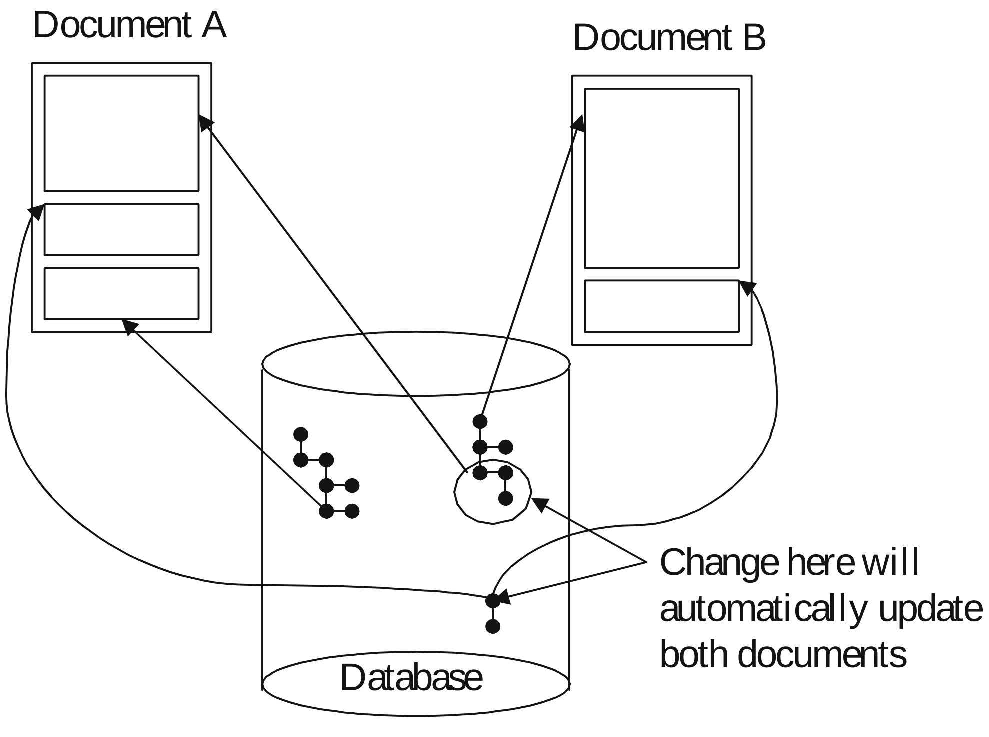


Figure 11: SGML Database.

1. For example Information Manager from Interleaf.

5.4 Writing Modular Documents

To get the best out of a document management system where multiple authors are editing the same document, or many documents have a lot in common with each other, it is best if parts of documents can be developed in relative isolation. This is common enough in software development, for example, where a problem is analyzed and different modules solve different parts of the whole problem. Modules usually have well-defined interfaces to other modules which makes it possible to develop the internals of the modules without knowledge of the internals of the other modules. It would obviously be cost-effective if traditional documents could be written this way as well, but, unfortunately, manuals are not computer programs. Differences in writing style, old habits and so on makes this very difficult.

Authors that have spent their whole career writing documents from start to finish by themselves are faced with the difficult transition of writing small document fragments, or micro-documents. The term micro-document can be a bit misleading, because it means that each document fragment describes a single component of some system, or a procedure. A micro-document for a bolt could be very simple, while a micro-document for a fuel system could be very large.

A micro-document should both be usable independently and it should also be possible to combine several of them into larger publications. This is difficult. Normal text cannot be detached from its context. Each chapter, for example, has first some introductory material and ends with some lead into the next chapter. This applies to the finer structure elements as well. Each paragraph starts with an introduction to the paragraph, and ends so that the next paragraph logically falls into place. If an automated document assembly process picks a piece from here and another from there, this old way of writing documents simply does not work.

Reality imposes some constraints on how document fragments are assembled together, of course, so writers are not left with an impossible task. For example, it could be known that every user manual will always have a safety section near the beginning of the document which is followed by a tools section. Also, it could be known that the documentation for some piece of equipment is only used in a certain product and not in any others.

Practicality has something to say as well. A document must naturally be readable. If an optimal solution would produce unreadable documents, larger blocks must be written. It may also be as simple as deciding that some documents will not be assembled automatically.

Structural documents are the natural way to write modular documents. When an organization decides to move into the structured document domain, there is usually a need to transform at least some of the old documents to the new format. Besides being a rather expensive and error prone operation, it may not be possible to split old documents into independent microdocuments at all. The transformations involved are outside the scope of this thesis, but they are discussed in [Tra95], for example.

5.5 Addressing External Resources

Managing blocks of documentation in SGML format is generally not enough. Most documents contain figures, some of which may need to be automatically generated at the time of printing. Static images (or video/audio formats) can be managed almost exactly like text documents. The difference is that they generally will be saved as large blobs of data and not broken down into smaller parts. Because it is more difficult to search images for certain topics, keyword and other metadata information is often attached to the binary format files when they are saved into a repository. For dynamic figures and tables there must be utility processes that can generate the data on demand.

Most technical documents include cross-references. SGML's limited abilities can be enhanced with other standards like HyTime, but that does not help in the management of cross-references. For example, how is an author supposed to make a cross-reference to a section that is the responsibility of another writer who has not yet begun his work? And what happens if document assembly process selects a piece of text for insertion into a publication and there is a cross-reference to another piece of text that is not included in the publication? Is it possible to automatically check that all cross-references point to valid targets?

The problems with link management warrants a thesis of its own. And the sad answer is that there is no perfect solution. Some rather simple techniques can be used to automate some tasks. Let's take a closer look at the three questions we have posed above.

It turns out the easiest problem to solve is linking to content that does not yet exist. This can be accomplished by creating a document that contains all the targets of links from the main document. These targets themselves redirect the link to its actual target (see Figure 12). This makes it possible for the first author to finish his part of the document before other parts even exist. Of course there must be some kind of an overview of the completed document so that it is possible to insert cross-references pointing to unfinished sections. During authoring, it does not really matter if the actual target does not yet exist because the intermediate document contains the information that some parts are missing and must be created before publication.

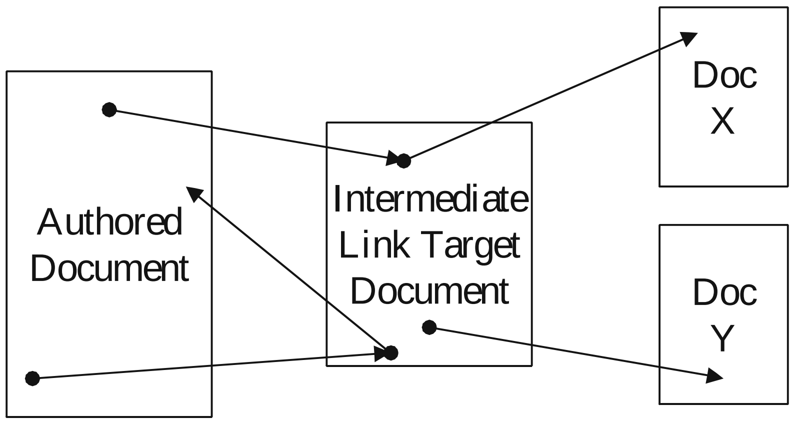


Figure 12: Authoring with an Intermediate Link Document.

The next problem is more difficult. If an automated document assembly process drops cross-reference targets off the publication, something must be done with the links or the document could become invalid or unusable. The simple approach is to use the intermediate link target file in this case as well. Links that point to non-existing targets simply point to information that explains that the link target is missing and where the missing part might be found. If the target is known to be in some other publication, the link can be replaced with a bibliographic reference to the other publication. In some simple cases cross-references can be deleted.

The most difficult problem is making sure that links point to where they are supposed to point. Links that point to relative locations may be confused by changing the document structure. For example, if a link points to the third paragraph of this section, deleting the current first paragraph changes the link target. The link might still be valid, or it might not. Although automatic processes can be created to check that every link points to something, a human is needed to make sure that the relationship is real and correct. There is no artificial intelligence system yet that could follow links and read or otherwise experience what is at the other end and reason out if the link was valid. It seems absurd that highly paid professionals sit for days at computer terminals clicking links and seeing where they lead to. But there is just no better way.

Problems and solutions in linking have been explored in [Kim98] and in summary format in [Ang97].

5.6 Client-Server Architecture

In a document management system, as described in this paper, there is a database and possibly some kind of file server. These are server-level components. The clients in a typical document management system will include editors or other data producers and viewer and publishing applications. A sample architecture is presented in Figure 13.

The client-server architecture model is very common way to design system architectures. It is a distributed system model that scales well from the needs of a single user to gigantic proportions. For example, the world wide web is based on the client-server model where clients (browsers) talk to web servers.

The architecture consists of three major components: server, client and network. The network component is optional. The server and client need to communicate with a predefined protocol. Changing the protocol at one end requires changes at the other end. Clients will also need to find the servers they are interested in.

The processing of information can occur centrally at the server, or the work can be divided between the server and clients. Generally it is better to have the server doing the bulk of the work because it is easier to maintain a few servers than several clients. However, as the number of clients increases, the server capacity and, in all likelyhood, the network capacity will need to be updated. The capacity of a single server can be improved by replacing it with a cluster of servers. The client continues to communicate with the cluster as though it were a single server. Inside the cluster the work load is balanced between different computers. This also improves fault tolerance, because failure in one computer does not render the cluster unusable. Using multiple servers, either in a cluster or more indepedently presents the problem that whenever data changes at one server it may need to pass this information along to the other servers.

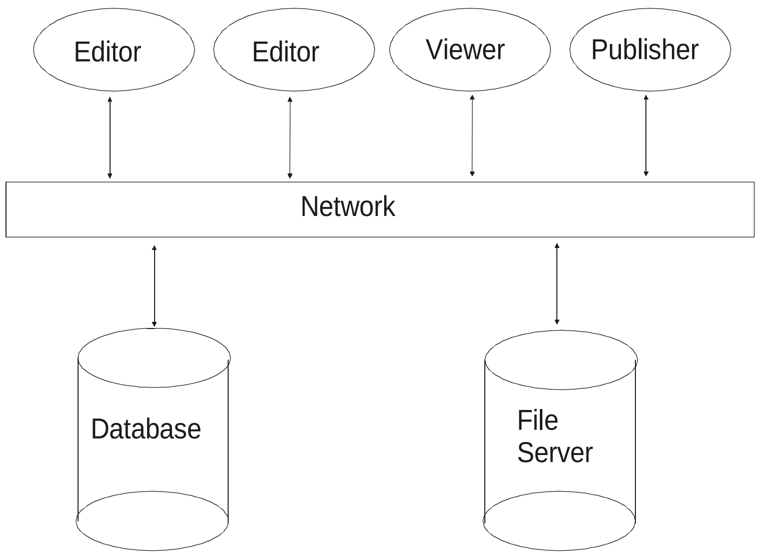


Figure 13: Client-Server Architecture for Document Management System.

5.7 Background Summary

This and the previous chapters have built the background knowledge needed to understand the practical part of this thesis. Product management, structured documents and databases have been explained with their strengths, weaknesses and practical considerations. This chapter in particular has shown what is involved in product document management when it is done with structured documents and databases, emphasis on relational database technology.

The following chapters introduce a real life product document management system using relational databases and SGML to store and manage product documentation. The system was implemented at Wärtsilä NSD by Citec Engineering Oy. The implementation does not use all of the features presented in this and previous chapters, but all of the features have been considered at some point. Other, similar systems have been described in [Tra95] and [Loo98]. The former briefly describes an SGML document management system using relational databases and the latter an image database implemented with the help of object-oriented databases.

Chapter 6

Product Information Management Project at Wärtsilä NSD Power Plants

The Duncans sometimes ask if I understand the exotic ideas of our past? And if I understand them, why can't I explain them? Knowledge, the Duncans believe, resides only in particulars. I try to tell them that all words are plastic. Word images begin to distort in the instant of utterance. Ideas embedded in a language require that particular language for expression. This is the very essence of the meaning within the word exotic. See how it begins to distort? Translation squirms in the presence of the exotic. The Galach which I speak here imposes itself. It is an outside frame of reference, a particular system. Dangers lurk in all systems. Systems incorporate the unexamined beliefs of their creators. Adopt a system, accept its beliefs, and you help strengthen the resistance to change. Does it serve any purpose for me to tell the Duncans that there are no languages for some things? Ahhhh! But the Duncans believe that all languages are mine.

Frank Herbert, *God Emperor of Dune*

Wärtsilä NSD (former Wärtsilä Diesel) is a Finnish engineering group with global operations. It is the leading supplier of power solutions for both land and sea. The gas and oil-fired power plant solutions range from 1 MW to 400 MW and are used for base load, co-generation, load management and gas compressor applications. The deliveries include turnkey construction and long term maintenance and operation.

Wärtsilä NSD Power Plants has about 300 active subcontractors, and has had over 8,000 different suppliers since 1982 [Pel97a]. Those active subcontractors are involved in over 100 power plant projects a year. These subcontractors are required to deliver to Wärtsilä documentation along with their products. The many different systems utilized by the subcontractors led to problems at Wärtsilä, and it was determined that imposing documentation standards on subcontractors would alleviate those problems. The Product Information Management (PIM) project was started to sort out the different problems and implement a solution. PIM was started out in 1995, while the majority of the work in the project was done during 1996 and 1997.

This chapter describes the overall PIM project and the project phases. The next two chapters describe two software products developed as part of the project in more detail. The author developed the second tool (see Chapter 8) as the practical programming work in this thesis.

6.1 Analysis Pointed to SGML And Relational Databases

It was determined that the different file format problems could be solved with SGML, so it was decided that all the subcontractors should supply the textual information in SGML. At that time SGML was still new in Finland: it was a rather radical solution for the time. The annual SGML Finland conferences had not even started yet, the first conference being held in 1996.

Wärtsilä was using Oracle relational databases internally, as were many of its larger subcontractors. Microsoft's Open Database Connectivity (ODBC) [Mic92] technology provided a strong reason to continue to favor relational technology as it allows applications to communicate with any ODBC-enabled database. SGML databases at the time were very expensive and not suitable for Wärtsilä's needs because the subcontractors were creating the documentation and they needed the document management system as well. Moreover, the PIM system was to store traditional product data in addition to being a document management system. It was seen that a new database schema would be needed to best utilize the system.

Content production was the next challenge, because most subcontractors were not using SGML internally. A survey was conducted among the subcontractors, which showed that the majority were using Microsoft products. To reduce the costs for subcontractors, and to make it easier for them to accept the movement to structured documents, a custom SGML authoring tool built around the Microsoft Word program was seen as the solution. The users would still be using the familiar Word program, there would just be new buttons and menu items. The authoring tool could be used to write SGML documents and at the same time, keep the document database up-to-date, and edit some database fields directly.

The final piece was the viewing and publishing tools. There were no good and cheap SGML viewers, not to mention specialized publishing packages, available and this meant that they would need to be created. In addition to being able to view and format SGML documents for display and printing, the programs would need to be able to communicate with the document repository to assemble larger works from the stored microdocuments.

6.2 Requirements And Specification

The PIM system requirements were loosely defined. Because the basic problem was that documents could not be produced in time, the main goal was to speed up the documentation processes. The use of SGML was seen as critical aspect to achieve this. For example, the authors would not need to spend time specifying styles, all necessary parts of documents would be created in the expected order and document assembly could be largely automated. It would also be easier to reuse information. The added benefit would, of course, be that document quality would improve.

Documentation is created at the same time as the new equipment it documents is manufactured. Occasionally snapshots of the current documentation are requested. It was expected that it would be easier to provide snapshots with the new system, and locate pieces of documentation that were not yet finished.

When Wärtsilä delivers a power plant, dozens — maybe hundreds of thick binder manuals are shipped to the customer. This takes a lot of space, is difficult to transport and is often difficult to get through customs procedures in several countries. Changes in documents can also require a lot of time to actually end up at the customer’s site. It was hoped that eventually paper manuals could be abandoned. Only CD-ROMs carrying the SGML files would have to be shipped to customers, and changes could either be shipped with more CD-ROMs or email.

SGML is a neutral data format in that it does not specify how it should be formatted, or even on what display system it should be displayed. It does not even need to be displayed at all, but could be read, or only manipulated by computer programs. This neutrality was also seen as important by Wärtsilä, because they could produce information in various formats generated from a central SGML source.

Software specifications were written first for the editor and database parts of the system, as well as for the plain SGML viewer program that would be used as the basis for the publishing tool. It could be argued that the specifications were no more than software definitions because they did not go into great detail about how the system should be implemented. For example, the documents stated that relational databases were to be used with the Open Database Connectivity interface, but the documents did not describe many of the dialogs that would be presented to the user nor were there detailed speed or memory usage requirements. The documents were mostly functional descriptions approved by the customer (Wärtsilä).

The natural development process for this project turned out to be evolutionary development. It was exploratory programming in the sense that the developers had to work with Wärtsilä to find out what the final system should be because of the loose definitions. It was exploratory programming also because the developers had to learn how to use some of the system components.

6.3 Design and Architectrure

The analysis phase had identified the key components in the PIM system. Thus the design process for the overall architecture and system data structrure design was relatively straightforward.

6.3.1 Architectural Design

The client-server model architecture was the natural choice. There is a server that hosts the relational database. The server also acts as a file server. The WNS Author Tool (described in Chapter 7) is used to create content and is one kind of client application. The publishing tool (see Chapter 8) is another client application. It is used to publish different information products (and “assemble products”) from the data repository managed at the server. The server and different clients could be located on the same computer. In fact, subcontractors would almost certainly have a mini-server sitting in their PCs. The mini-server would have only a part of the information contained in the main server located at Wärtsilä.

The overall system architecture can be seen in Figure 15 (see a reduced view in Figure 14). Creation of Content represents the WNS Author Tool, Document Management System is the server and Product Assembly and Formatting Application is the publishing tool (Multidoc Pro Database Publisher) that was developed as the practical work for this thesis by the author.

There was never any question as to the database technology to use. Wärtsilä had all the product data in an Oracle (relational) database. The system was to be built around it. However, because the relational database stores only references to files the reference could be to an SGML document held in a special purpose SGML database.

The subcontractors were to transfer the files using FTP to Wärtsilä, including their light version LSAR (see Section 6.3.3). A workflow system would initiate a workflow upon delivery of the files to the Wärtsilä server. The files were to be decrypted and scanned for viruses, after which they should go through the approval processes at Wärtsilä. Documents that were not approved would be returned to subcontractors for more work while approved information would be saved to the main data repository at Wärtsilä. Tight integration of the database and other system components was part of the overall vision, but it was not planned for initial implementation.

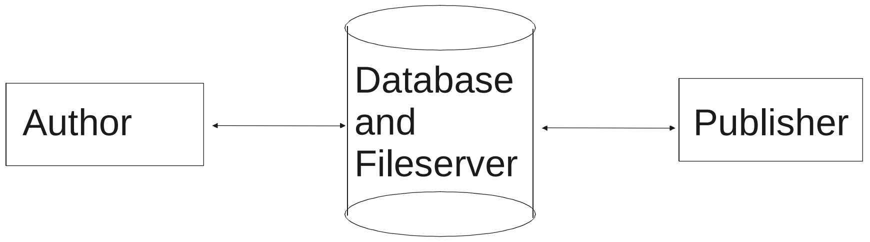


Figure 14: Simplified PIM System Architecture.

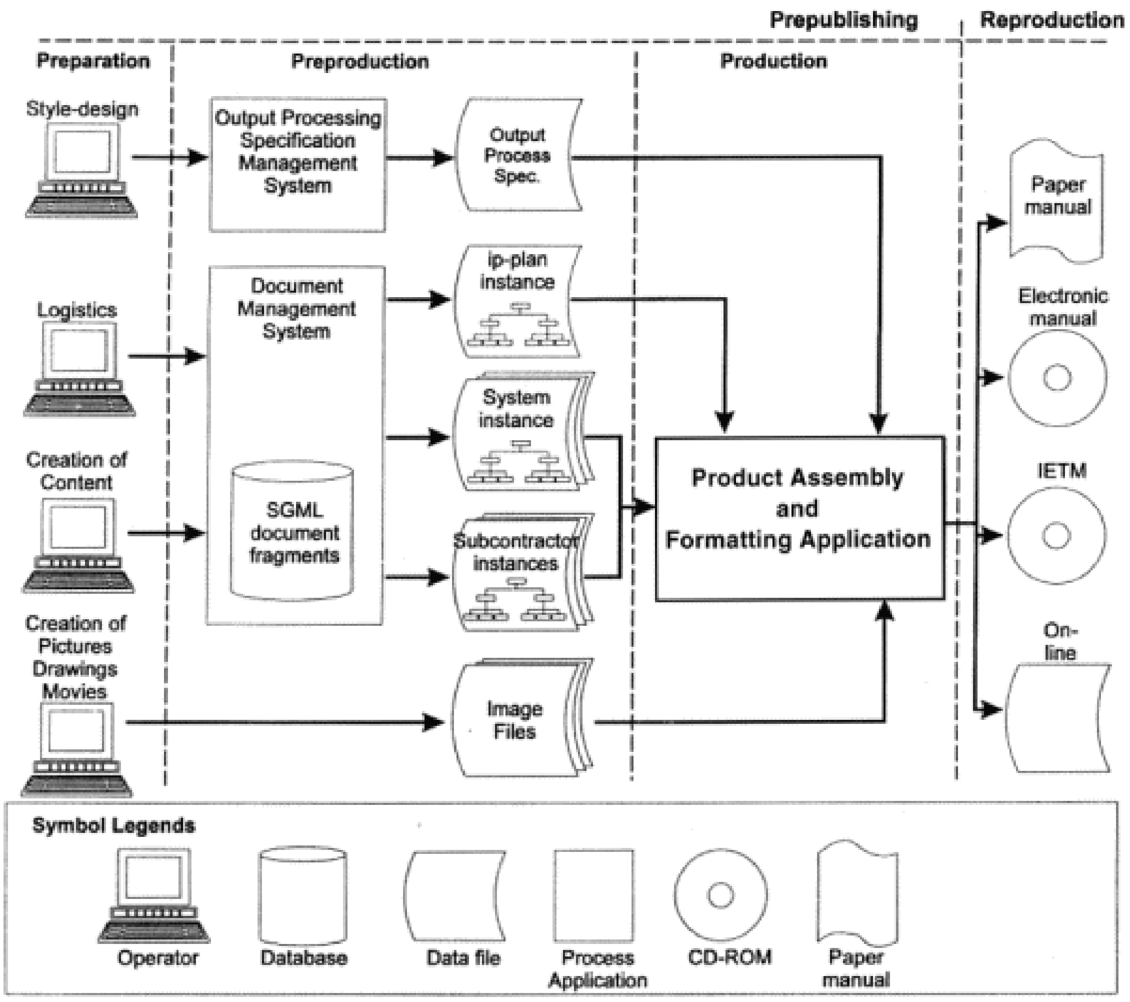


Figure 15: System Architecture.

6.3.2 DTD Design

System-level data structure design involved designing SGML DTDs and the database schema. Citec[1], subcontracted by Wärtsilä to design and implement the whole system, developed eight different document types for the documents needed at power plants. The FMV DTD (designed for and used by the Swedish military and subcontractors) was used as the model for these DTDs [FMV95]. FMV is content-oriented (i.e., it uses logical element names that describe what the data is as opposed to structural names like chapter and section), which was exactly what Wärtsilä wanted. The full FMV DTD has a lot more “branches” than the eight that were selected and modified for Wärtsilä, but it was decided to start simple and later, if needed, integrate the rest of the FMV DTD.

1. Citec is the largest SGML service provider in Finland. The company homepage is `http://www.citec.fi`. The author was hired by Citec in the summer of 1996.

The eight DTDs that were designed are: system, function, operation, corrective maintenance, periodic maintenance, technical data, faulfinding and spare parts. Figure 16 shows one of them, the spare parts DTD in tree view. The DTDs are documented in [CIT97b].

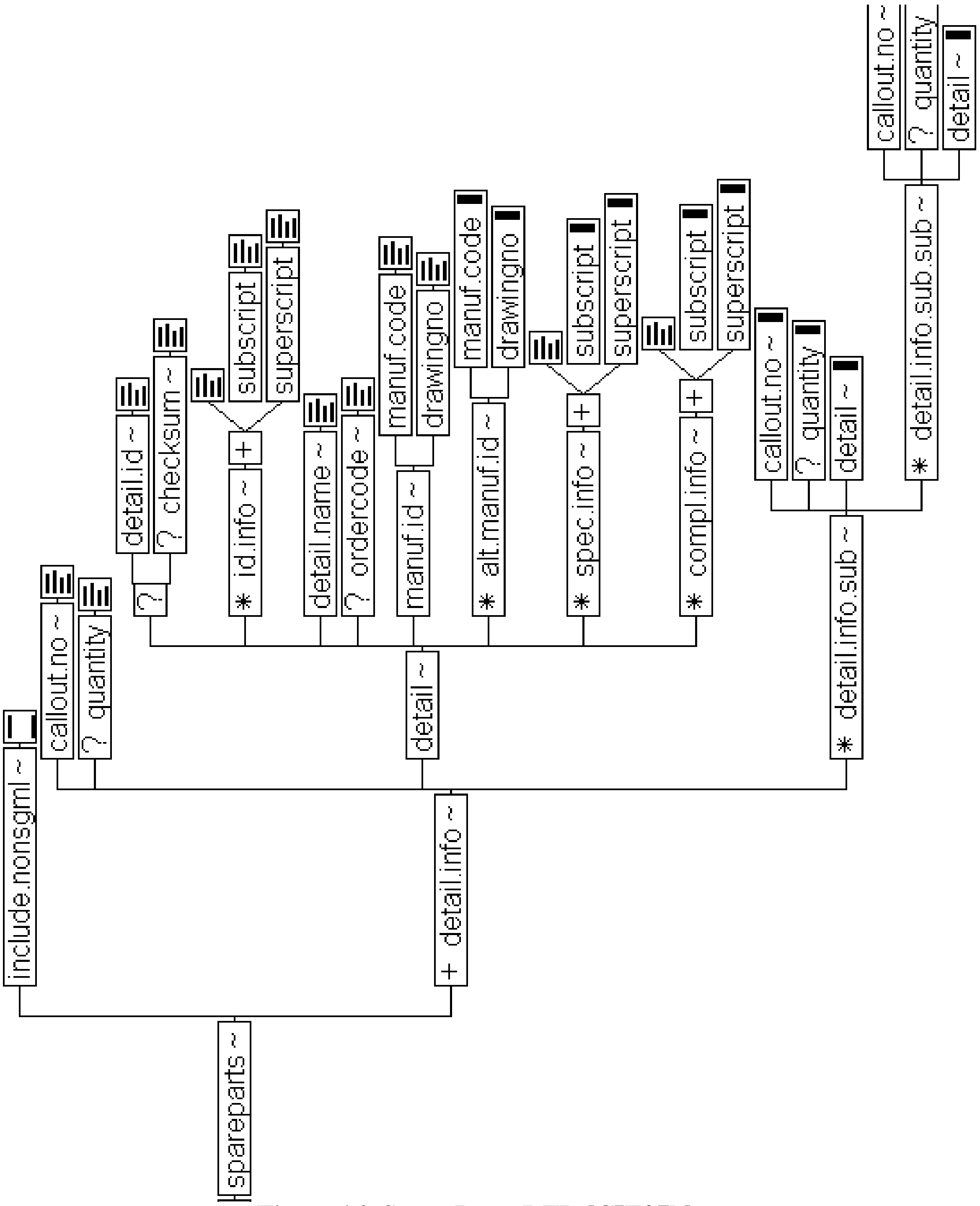


Figure 16: Spare Parts DTD [CIT97b].

If and when these DTDs get revised, it will be a lot easier to migrate the old data to conform to the new specifications relative to the old situation where information was not in a standardized format. Still, transformation is not a trivial problem (see for example [Lin97]).

6.3.3 Database Design

A logistics support database (LSAR) — or Equipment Breakdown Structure (EBS) as it is described in the reference — holds the logistics information about different components and their documentation. Images in various predefined formats are also managed by the database. The LSAR is a normal relational database. The authoring tool creates SGML files and keeps the local LSAR database up-to-date. EBS is like the logical view of the LSAR database. Figure 17 shows the EBS view.

The final LSAR database schema is shown in Figure 18. Because of the tree-like structure of the EBS, the database also simulates a tree-like structure. This is achieved with the `Parent ID` column in the `Structures` table and the `Parent Code` column in the `PPS Codes` table. For example, in the EBS we can see that a Fuel System consists of at least Oil Heater and Ball Valve. Those components must have Fuel System as their parent. The components shown in the EBS are generic types of components and they are listed in the `PPS Codes` table. That, and other tables, are explained below.

The `Structures` table holds information about each individual structural component in a power plant. `Parts` contains information about spare parts, and the `Documents` table stores information about documents. Other tables are more or less auxiliary tables that were created while optimizing the database. An earlier version of the database can be seen in Figure 32.

The `PPS Codes` table lists the generic types of components in a power plant. Each generic component can have documentation associated with it, via the `PPS Documentation` table. The `Serial Number` column in the `Structures` table shows that it is used to track additional information about each manufactured component. Thus, there can be different documents for the same kind of fuel pump, one of which is installed in a power plant in Beijing and one in Ankara, for example. The `Structures` table connects to the `Documents` table via the `Structure Documentation` table. Of course, there must also be a relationship with the `PPS Codes` and `Structures` tables, because each individual component is always of some generic type of a component.

There are many components in a power plant that need maintenance after a certain amount of time. Obviously there must be documents that describe how these maintenance operations are to be carried out. The `Per Maint Intervals` table is a helper table that makes it possible, for example, to search documents for maintenace operations held every 500 hours.

The LSAR database is at least in the third normal form. It has not been checked to see if it would qualify for more advanced normal forms.

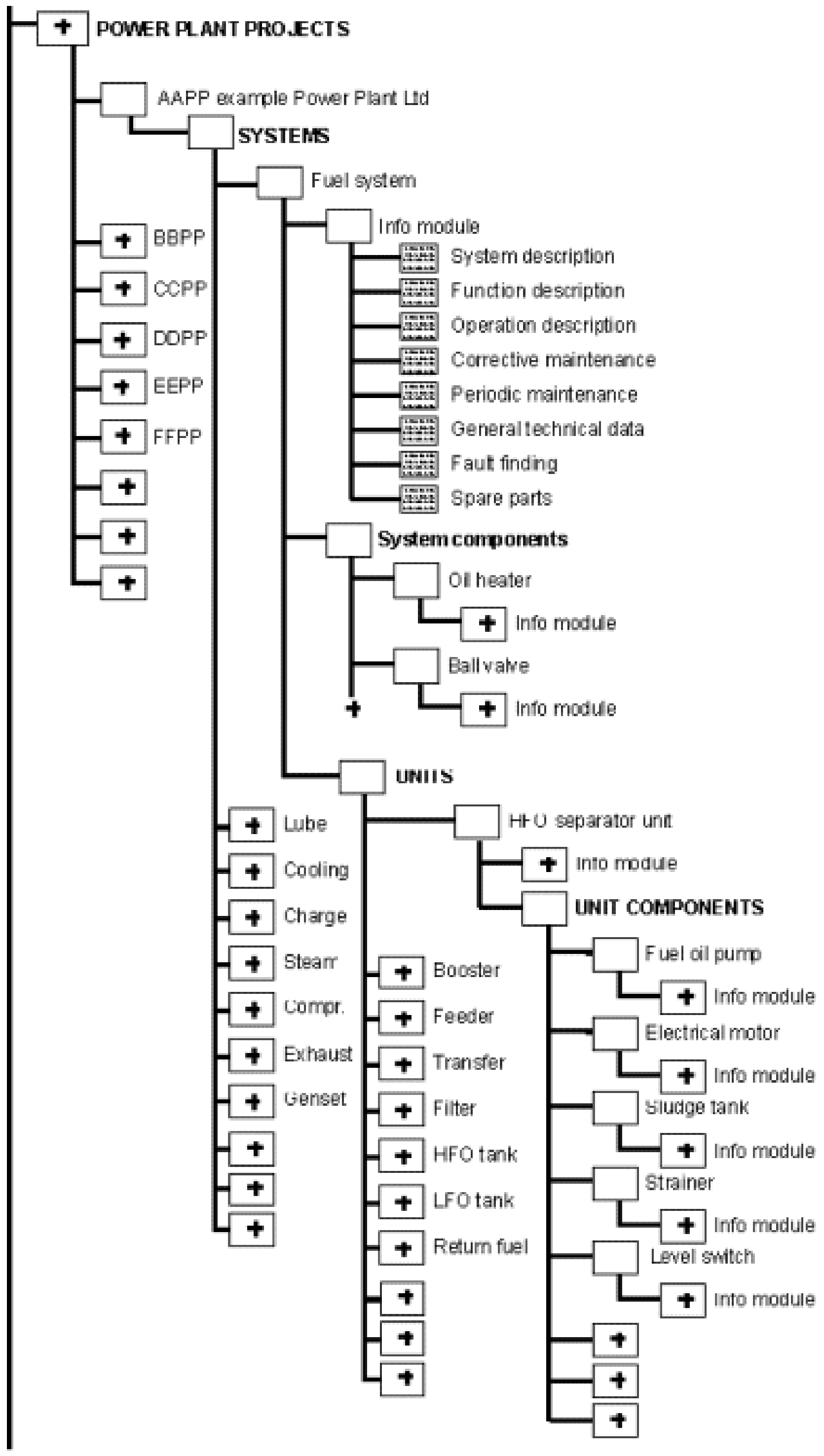


Figure 17: Power Plant Equipment Breakdown Structure.

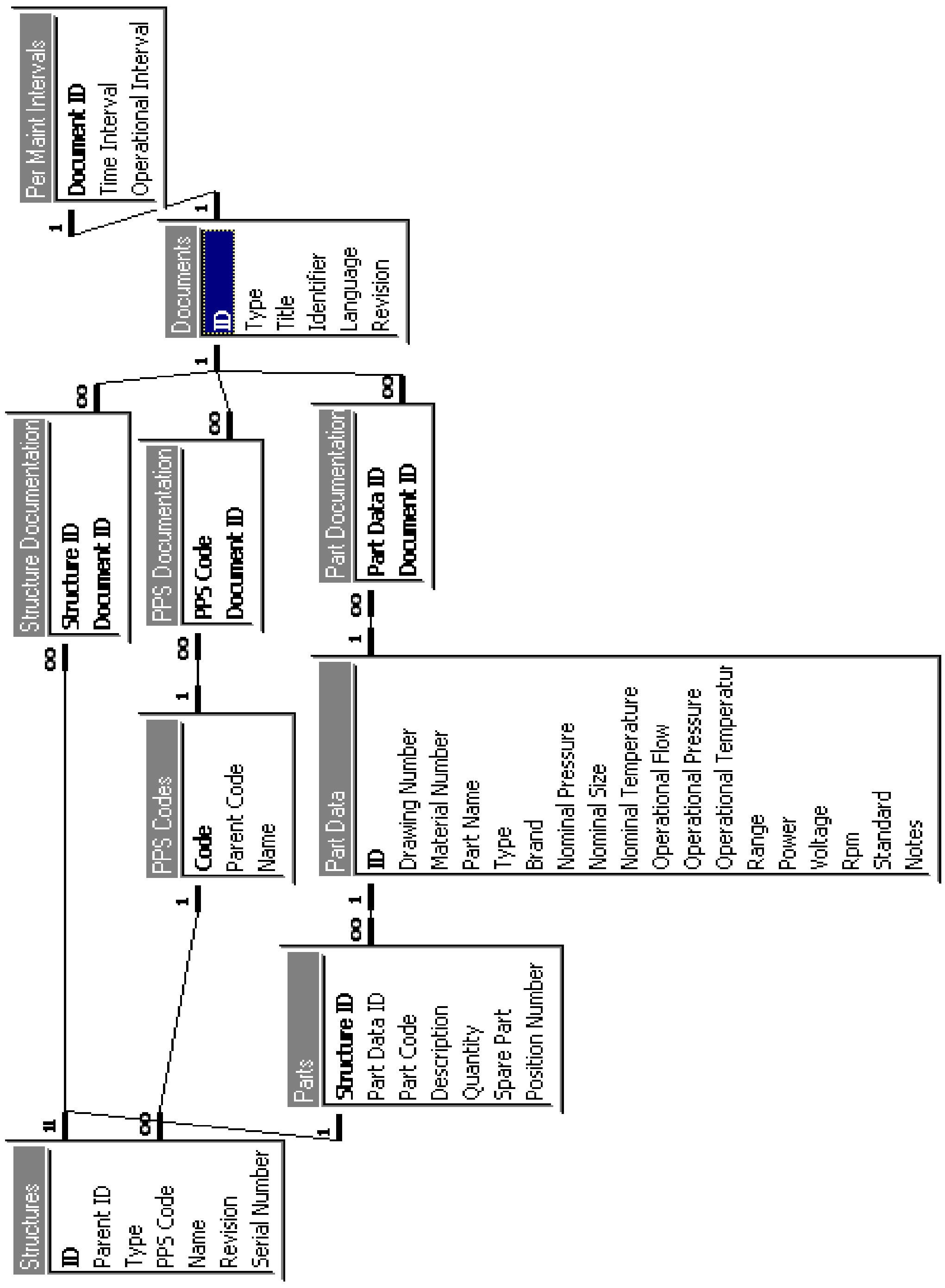


Figure 18: LSAR Schema.

6.4 Verification and Validation

No metrics were designed to measure the success or failure of the delivered system. Because of the loose definitions it would have been quite difficult to measure anything. In retrospect, it can be seen that at least one thing could be clearly measured: what percentage of documentation projects are completed on time. Additionally, user satisfaction could be measured with interviews. As the system has many different user groups, ranging from authors to end product users, several group satisfaction profiles could be created. In fact, there were was a preliminary plan to observe the system and its users in a controlled environment. There was an attempt to get external funding for this testing, but it did not work out and the plan was abandoned.

Although the tools more or less did what they were supposed to do, the overall project failed at Wärtsilä. There were several reasons. The transition to SGML was not properly orchestrated. The project did not gain company-wide acceptance. It might have helped if the tools would have exceeded expectations, but they took longer to develop than anticipated and were not as good and easy to deploy as was hoped.

In retrospect it is quite easy to see what should have been done differently. The whole project at Wärtsilä should have been handled differently. Acceptance at all levels is crucial for a transition to SGML to succeed, and more people should have been involved. The SGML project at Wärtsilä did not get any extra resources to carry out the testing and integration of the new system. The authors and editors were expected to deliver information products using the existing system and simultaneously install the new system, evaluate it, provide feedback to Citec and begin using the new system in earnest.

After evaluation of the Microsoft Word-based editor the vision should have been re-evaluated. The originally chosen solutions were proved limited. The correct decision would have been to make a fresh start with a native SGML editor (for more details see Section 7.2).

It was also learned that the subcontractors were not ready to move into SGML. The jump to structural information was simply too great, even with all the preparation and tools provided.

Wärtsilä and Citec have evaluated the system using information prepared for the Wärtsilä Pilot Power Plant located in Vaasa. Other small scale testing and use has been carried out by Wärtsilä personnel. Perhaps the biggest value in the project has been to gain knowledge of SGML and everything involved. The current SGML documentation projects are using the knowledge gathered from this project, and they seem to be faring better.

Chapter 7

Document Authoring

AXIS (Biologic Band 4)> Hello, Roger. I assume you're still there. This distance is a challenge even for me, based as I am upon human templates... [politeness algorithm diagnosis for total mechanic-biologic thinker function V-optimal] most of the time. I have come within a million kilometers of B-2 mark this moment 7-23-2043-1205:15. I am preparing my machine and bio memories for receipt of information from the children, now flying in a perfectly dispersing cloud toward B-2. Data on B-3 have been relayed. The planet, you can see, is quite Jovian, very pretty, though tending towards the greens and yellows rather than reds and browns. I'm enjoying the extra energy from B's light: it allows me to get some work done that I've been delaying for some time, opening up regions of memory and thought I've closed down during the cold and dark. I've just completed a self analysis; as you doubtless have discovered by checking my politeness algorithm diagnostic. I am V-optimal. I am not using the formal "I"; the joke about self awareness still does not make any sense to me.

Greg Bear, *Queen of Angels*

There are several good SGML editors on the market, for example Adept•Editor. Most of them are quite expensive, and require mastering operation concepts which have little in common with other types of software. Obviously, they are general purpose tools. The requirements for the Wärtsilä project made the development of a custom authoring tool necessary. The primary source of information for this chapter was [CIT97b].

7.1 Authoring Tool

The WNS Author Tool created by Citec is used to “enter database records and write corresponding SGML documents according to the WNS Base-DTD”. Simply put,this means that by selecting a component from the Equipment Breakdown Structure (EBS) (see Figure 17 in Section 6.3) one can write documentation for that specific component. The EBS information is in the LSAR database. Information about the documentation modules are also saved into the LSAR database.

The authoring tool works with Microsoft Word 6. SGML Author for Word 1.0 is also needed, along with Microsoft Access (database) Driver.

A new document is created by selecting `New` command from the `File` menu and selecting the LSAR template. The default view is presented in Figure 19. In the figure the area marked with 1 on the left contains the editing fields. Area 2 contains the number and the type of information modules contained in the record. Scrolling buttons are located in Area 3, and Area 4 is the database display field. The next step is connecting to the LSAR database. This is accomplished simply by double clicking the connection button (not visible in the figure), and selecting the LSAR data source.

Changing the context in the EBS is accomplished via the context selection dialog (see Figure 20). Editing fields are filled from drop down lists (see Figure 21). Information modules are attached by double clicking one of the eight module buttons (not visible in the figures), for example, `Operation`.

Information modules are written, more or less, as normal documents with Word. The SGML structure must be generated in the import and export phases from style information because Word does not handle it natively. Therefore, it is very important that only the styles defined in the LSAR and information module templates be used. A sample information document can be seen in Figure 22. The styles from which the SGML information will be generated are visible on the left.

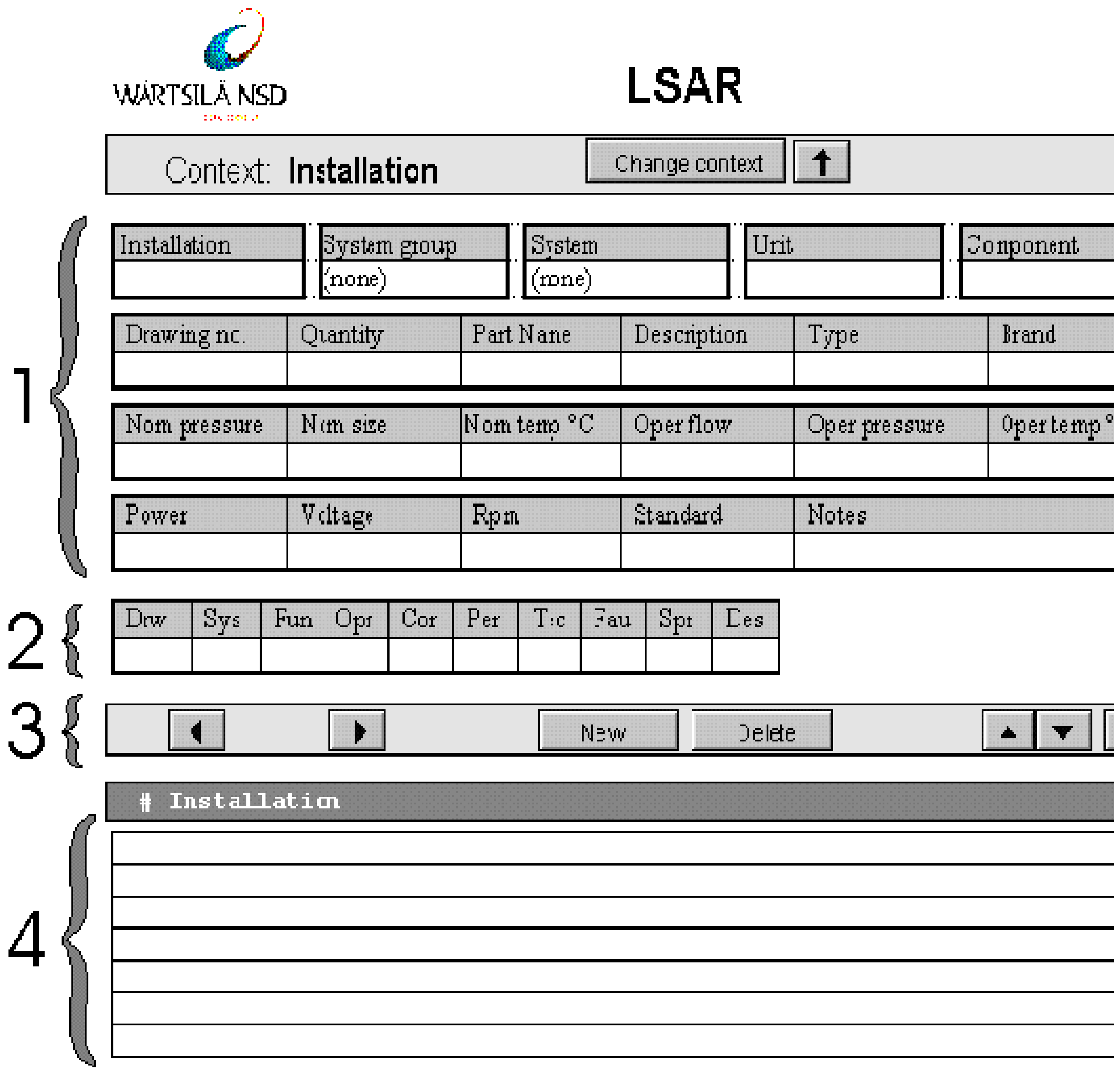


Figure 19: The LSAR Interface.

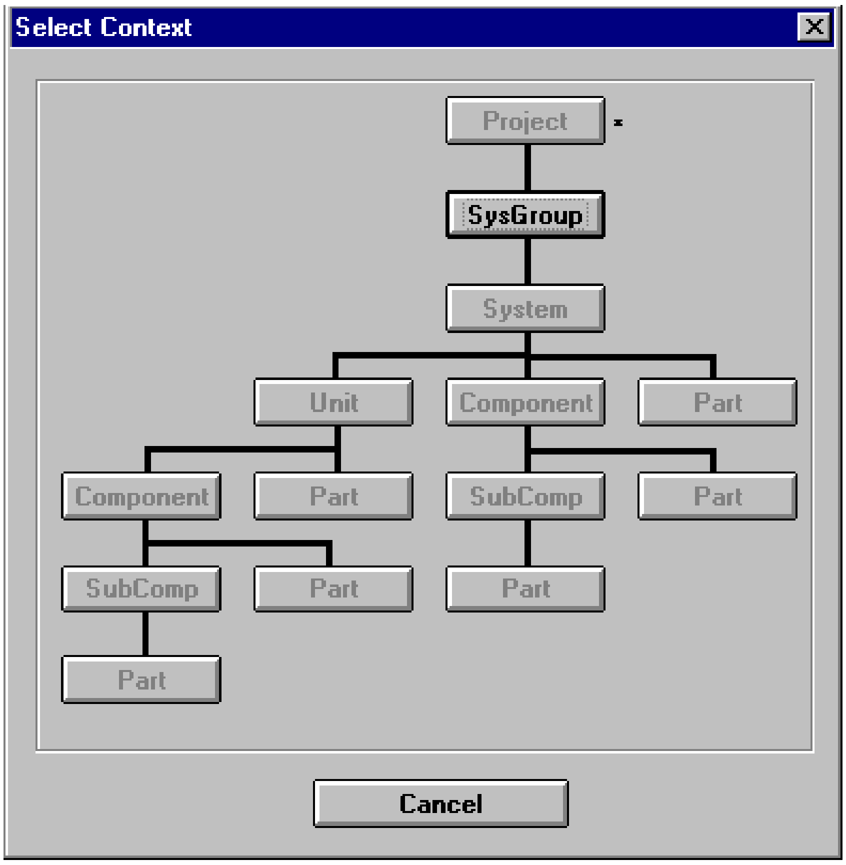


Figure 20: Selecting Context from the Equipment Breakdown Structure.

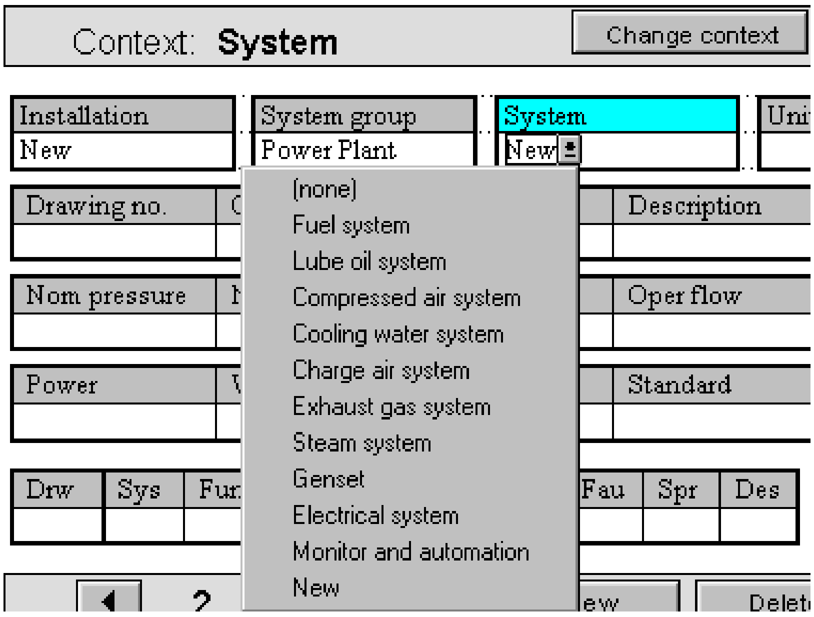
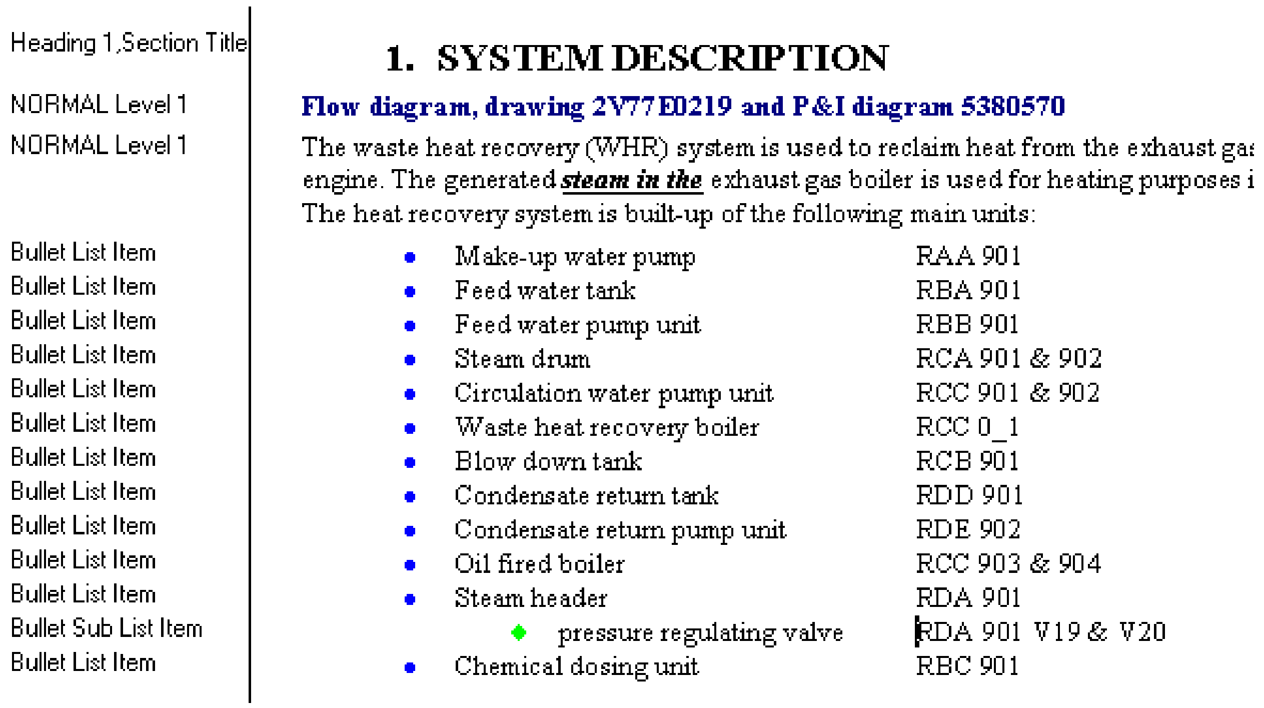


Figure 21: System Field Drop Down Menu.



Heading 1,Section Title	1. SYSTEM DESCRIPTION
NORMAL Level 1	Flow diagram, drawing 2V77E0219 and P&I diagram 5380570
NORMAL Level 1	The waste heat recovery (WHR) system is used to reclaim heat from the exhaust ga engine. The generated ~~***steam in the***~~ exhaust gas boiler is used for heating purposes i The heat recovery system is built-up of the following main units:
Bullet List Item	• Make-up water pump RAA 901
Bullet List Item	• Feed water tank RBA 901
Bullet List Item	• Feed water pump unit RBB 901
Bullet List Item	• Steam drum RCA 901 & 902
Bullet List Item	• Circulation water pump unit RCC 901 & 902
Bullet List Item	• Waste heat recovery boiler RCC 0_1
Bullet List Item	• Blow down tank RCB 901
Bullet List Item	• Condensate return tank RDD 901
Bullet List Item	• Condensate return pump unit RDE 902
Bullet List Item	• Oil fired boiler RCC 903 & 904
Bullet List Item	• Steam header RDA 901
Bullet Sub List Item	◆ pressure regulating valve RDA 901 V19 & V20
Bullet List Item	• Chemical dosing unit RBC 901

Figure 22: Sample Information Module in WNS Author Tool.

7.2 Implementation of the WNS Authoring Tool

The first implementation work in the PIM project begun with the development of the authoring tool. The first specifications were too optimistic; a functioning editor was expected in less than six months. In reality, it took well over a year to ship the final version, although not all of that time was spent on the authoring tool development alone. There was one developer allocated to working with the authoring tool and the LSAR database schema.

The WNS Authoring Tool was implemented in Microsoft Word Basic. This presented some serious difficulties in programming. For example, the Word Basic application programming language only allows function calls to nest five levels deep in other macros.

The authoring tool and the LSAR database were tested by Citec and Wärtsilä by trying to create documentation with it. There were no software tools used in the testing. There was not really any viable software available that could have been used, apart from test suites that actually run the program and record inconsistencies between runs. This kind of software is expensive, and was not seen to be cost effective.

The biggest problem with the Authoring Tool is that it is limited to a specific version of Microsoft

Word and SGML Author for Word. As newer versions of Word emerged, users of the Authoring Tool were locked to version 6, because it is, for all practical purposes, impossible to use multiple versions of Word on the same computer.

As soon as this was realized plans for improvement were prepared. In these plans the parts that offer connection to databases and the specialized knowledge about the eight DTDs would be coded into a generic DLL. After that, any SGML editor could be customized to write power plant documentation by coding simple hooks from those applications into this DLL. The plan was never carried out because the whole SGML project suffered problems at Wärtsilä.

An additional problem with the SGML Author for Word was licensing. It was not clear who owned the product and who could give licenses.

Citec's experience with other customized SGML editors suggest that it would not be a big risk to choose a native SGML editor like Grif[1] over a mixed implementation like SGML Author for Word. The benefits of native SGML will come with a more reliable environment and faster applications. Additionally, native SGML editors often have a rich API with the possibility of using real programming languages to implement additional packages. These outweight the initial, minor difficulties the users are faced with while learning a new program.

1. Grif SGML Editor is a product of Infrastructures for Information. Citec has created a customized SGML editor from Grif for the offshore industry (`http://www.citec.fi/company/products/toolbox.html`).

Chapter 8

Document Assembly

Inaccuracy. We did not destroy those portions of your organic brain. We borrowed/took/expropriated a few grams of tissue for use in a great goal. Our need was greater than yours'.

David Brin, *Heaven's Reach*

The practical work in this thesis was concentrated into the area of document assembly. See Figure 15 for how this relates to the Product Information Management project at Wärtsilä.

The work included expanding the functionality of the Multidoc Pro (MDP) SGML Browser so that it could be used to connect to relational databases and create compound documents from several document fragments managed by databases. In the Wärtsilä project the document fragments were authored with the tool presented in the previous chapter, but the use of that particular tool is not a requirement.

Additionally, Multidoc Pro was enhanced so that it could view relational databases as if they were SGML documents.

8.1 Multidoc Pro SGML Tools

Multidoc Pro[1] is the brand name for a variety of SGML tools based on the popular Synex View-Port SGML engine. The first tool was Multidoc Pro Browser, which was released in December 1996. Other versions soon followed.

The current tools in the Multidoc Pro product family are: Browser, Publisher, Database Browser, Database Publisher, Translating Editor, CD Browser and Plugin. A special internatiolization package for Multidoc Pro has been developed to enable translation of the program to different languages. These translations can be performed by anyone without the need to recompile the program. Several customized versions for different companies exist as well; for example, the Norsk Hydro versions of the Browser and Publisher support special HyTime contextual link (`clink`) handling.

Multidoc Pro did not grow out of nothing. Its predecessor was Multidoc LT, created for Wärtsilä Diesel. The LT also had a predecessor, Eldoc. Both Eldoc and Multidoc LT are based on different technology than Multidoc Pro and require that the SGML material be compiled to a proprietary format before it can be used. Multidoc Pro, on the other hand, is a native SGML product which does not require precompilation. Multidoc Pro is being developed with Microsoft Visual C++, while Multidoc LT was done with Asymmetrix Multimedia Toolbook. C++ enables much finer control over the program, not to mention the runtime speed benefit.

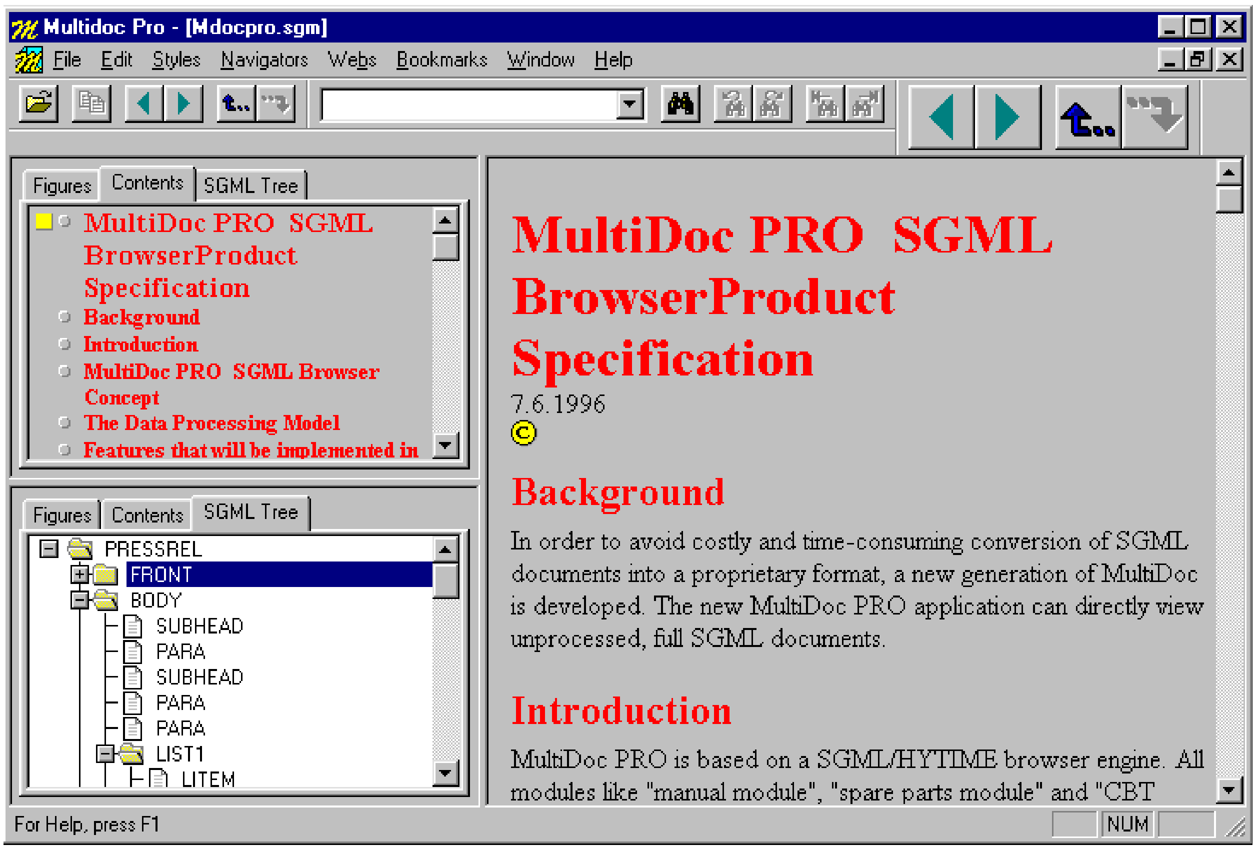
Some properties are common in all Multidoc Pro products. When an SGML document is opened in Multidoc Pro, it displays a document window and may be configured to display other windows (see Figure 23, right hand side). It is possible to define **navigators** for individual documents or define navigators for a specific public identifier. A navigator is like an electronic table of contents. The navigators are displayed to the left of the actual document window (see Figure 23). It is possible to define multiple navigators for a document, for example, a list of figures and a list of tables. The navigator can display graphics as well as text, so a list of figures can be very descriptive. In the navigator specification file the elements that one wants to appear in the Multidoc Pro navigator are specified with SGML.

Multidoc Pro formats the SGML instances before displaying them on the screen (or sending them to the printer). This formatting information is saved in special **stylesheets**, which are SGML files themselves. A document can have multiple stylesheets attached, although only one is active at a time.

A document can also have multiple **webs** attached to it. Webs are also SGML documents like navigators and stylesheets. Webs enable the annotation SGML documents, but a more exciting feature is provided as well — user defined links. This means that a reader can insert new links

1. Multidoc Pro is a registered trademark of Citec Engineering Oy.

between parts of the documents in addition to those created by the original author. The support for navigators, stylesheets and webs are all enabled by the ViewPort SGML engine, so those constructs created with Multidoc Pro work as well with any other ViewPort-based product and vice versa.


Multidoc Pro - [Mdocpro.sgm]
File Edit Styles Navigators Webs Bookmarks Window Help
Figures Contents SGML Tree
For Help, press F1
NUM

MultiDoc PRO SGML BrowserProduct Specification
Background
Introduction
MultiDoc PRO SGML Browser Concept
The Data Processing Model
Features that will be implemented in

PRESSREL
FRONT
BODY
SUBHEAD
PARA
SUBHEAD
PARA
PARA
LIST1
LITEM

MultiDoc PRO SGML BrowserProduct Specification

7.6.1996

©

Background

In order to avoid costly and time-consuming conversion of SGML documents into a proprietary format, a new generation of MultiDoc is developed. The new MultiDoc PRO application can directly view unprocessed, full SGML documents.

Introduction

MultiDoc PRO is based on a SGML/HYTIME browser engine. All modules like "manual module", "spare parts module" and "CBT

Figure 23: Multidoc Pro Screenshot.

One common use for the web files is document update. For example, if a company issues four CDs a year but would like to notify customers of changes between issues, web files can be created and emailed to customers. Customers then just attach the web files to the documents and immediately see where and what the changes are.

A construct that is not available in other ViewPort-based products is the document set. A document set is a HyTime document, with HyTime links pointing to the "actual" content (see Figure 24). Multidoc Pro offers a special document set navigator for document sets.

Multidoc Pro supports a wide range of graphics formats, starting from commonplace WMF, GIF, JPEG and bitmaps to somewhat more exotic CGM, PCX and PNG, for some example. Multidoc Pro has multimedia support — it can display multimedia files in any format provided the needed drivers are available. Common video formats include MPEG and AVI. Plain sound is naturally available as well. All graphics and multimedia objects can be shown inline (multimedia objects have the accompanying controls displayed with them, for example, the "play" button). It is possible to specify external helper applications for non-SGML data that Multidoc Pro can not show itself.

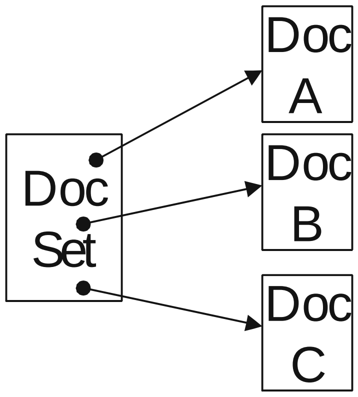


Figure 24: Document Set.

Although Multidoc Pro does not offer full HTML support, it can still function as a web browser. Naturally, SGML files can also be viewed over the Internet. The combination of a web browser such as Netscape Communicator and the Multidoc Pro Plugin provides a complete HTML and SGML Internet solution.

The Multidoc Pro programs are available for free evaluation period of 21 days from the Citec Web service. After the evaluation period a license key must be purchased from Citec Software Ltd. Unauthorized usage is prevented with CrypKey[1]. Multidoc Pro requires that a special Crypkey service is running in the computer (or in a computer connected to the network). The service is included in the installation program

8.2 Multidoc Pro Database Browser and Publisher

The database extensions to Multidoc Pro Browser and Publisher makes it possible to browse relational databases as if they were SGML documents. More importantly, it is possible to assemble large document collections or publications from several smaller SGML files that are managed by relational databases. Before the databases can be used with Multidoc Pro Database Browser or Publisher, the database structure must first be mapped into a database mapping file that Multidoc Pro understands.

1. See `http://www.crypkey.com` for more information.

8.2.1 Database Mapping

A database mapping starts by selecting ODBC data source names to connect to. A mapping can include many data sources simultaneously, although only one data source will be used for one dynamically generated SGML document. For each data source name in the configuration a mapping will be prepared. The mapping is simply a way to tie the hardcoded database DTD elements to a given database's structures. This is similar to the model-driven mapping described in Section 5.1.2.

A Data Source Name (DSN) is a concept in ODBC. It is closely related to the SGML concept of the Public Identifier as it is simply a symbolic name associated with a real address. All DSNs are registered with the ODBC driver manager. The driver manager knows where the symbolic name really points to and can therefore act as an invisible bridge between applications and databases.

The hardcoded database DTD (see Appendix A) in Multidoc Pro Database Browser and Publisher is very generic (yet simple, which is why it does not follow any methodologies for DTD structure, like the one presented in [Mal95]). It has a container for a table and a query, and that container has child elements that are used to output the different output columns of the query (or simply table rows if the container element is mapped to a table). The table or query container element is called `level`, while the different column elements are called `title`, `dataname`, `datadescription`, `reference` and `ref.name`. The `reference` element is a special element because it is used to create a HyTime link to an external SGML document.

Figure 25 shows a graph view of the DTD. The ASCII representation is in Appendix A. The figure is read from left to right and from top to bottom. The symbols are the same as in a normal DTD (see Section 3.3.3). The question mark (`?`) means an optional element. The tilde (~) means the element has attributes (attributes are not visible in the diagram). The connector between element name boxes specifies how the elements appear in the content model. All but `datavalue` have the same connector — this means that the element must appear in the order read from top to bottom. For example, the element `data` has an optional `dataname` followed by zero or more `datavalue`s. The `datavalue` element has the `OR` connector. Its content can be either zero or more `datadescription` elements or zero or more `reference` elements.

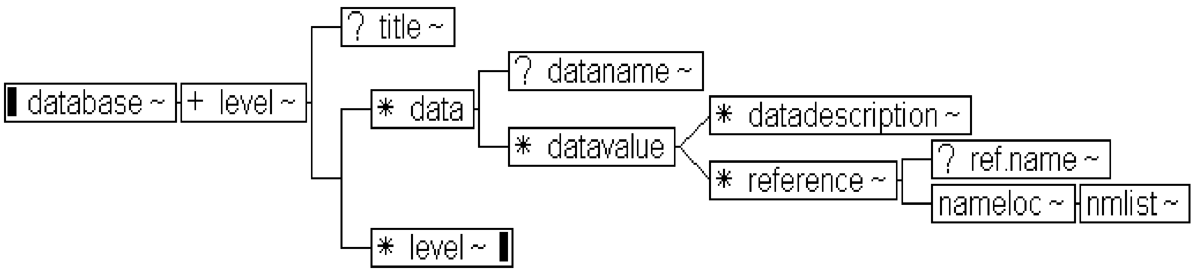
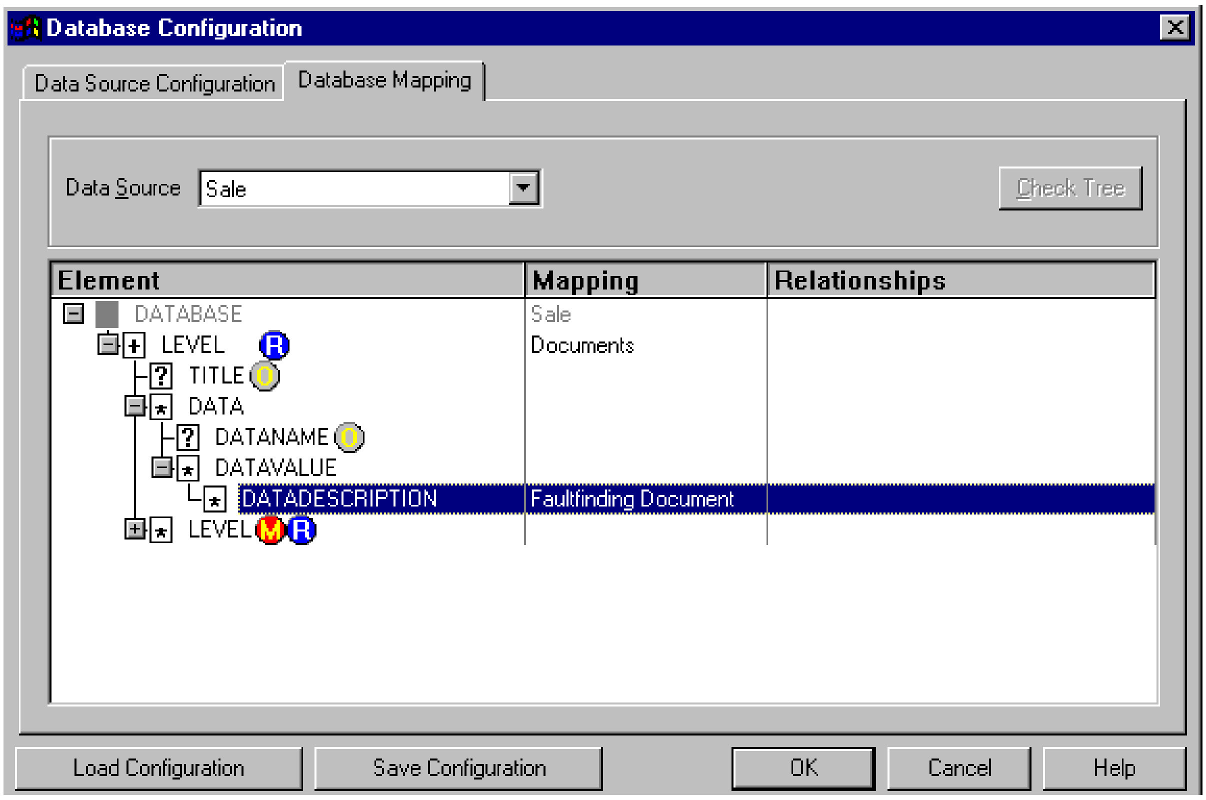


Figure 25: The Tree View of the Database DTD [CIT97a].

A mapping can be saved. The save file is in ASCII format and, unfortunately, not too yeasy to edit. The initial idea was to make the file binary, but ASCII was first chosen for debugging purposes. SGML would be ideal save format, but it has not been implemented yet. An advanced database DTD is being developed which will allow saving the whole mapping information in SGML format. This DTD can be seen in Appendix B.

The Multidoc Pro dialog where the mapping is specified is shown in Figure 26. Right-clicking on element names produces a context menu that has all the functionality needed to create the mapping (see Figure 27). The `Map Database...` menu item produces the dialogs `Map Table`, `Map Query` or `Map Column` depending on context. The dialogs are shown in Figure 28, Figure 29 and Figure 30, respectively. The `Map Query` dialog was added late in the project and is very crude in design compared to the other dialogs. `Relationships...` menu item opens the `Map Relationships` dialog (see Figure 31). All the items in all the dialogs have the standard Windows tooltips (mini-help windows that pop up when the mouse cursor hovers over a control) associated with them. All the screenshots are from [CIT97a].



Element	Mapping	Relationships
DATABASE	Sale	
LEVEL R	Documents	
TITLE O		
DATA		
DATANAME O		
DATAVALUE		
DATADESCRIPTION	Faultfinding Document	
LEVEL M R		

Figure 26: Database Mapping Dialog.

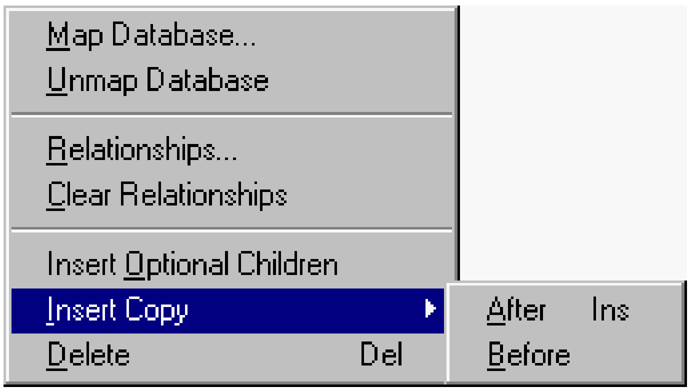


Figure 27: Database Mapping Context Menu.

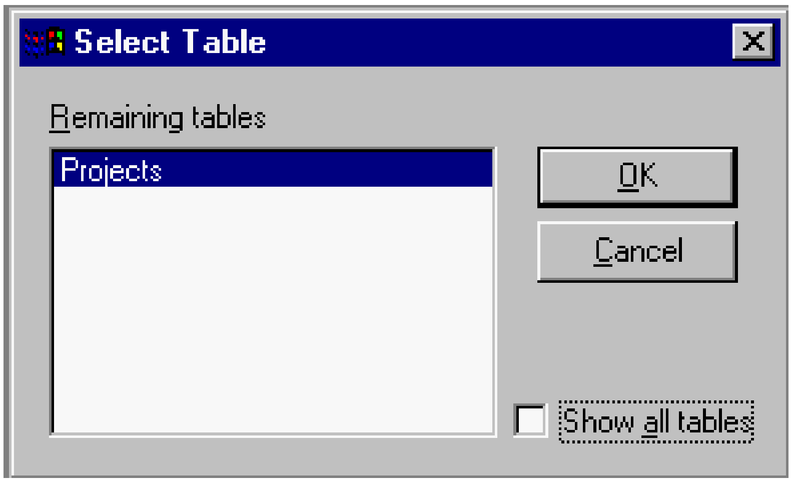
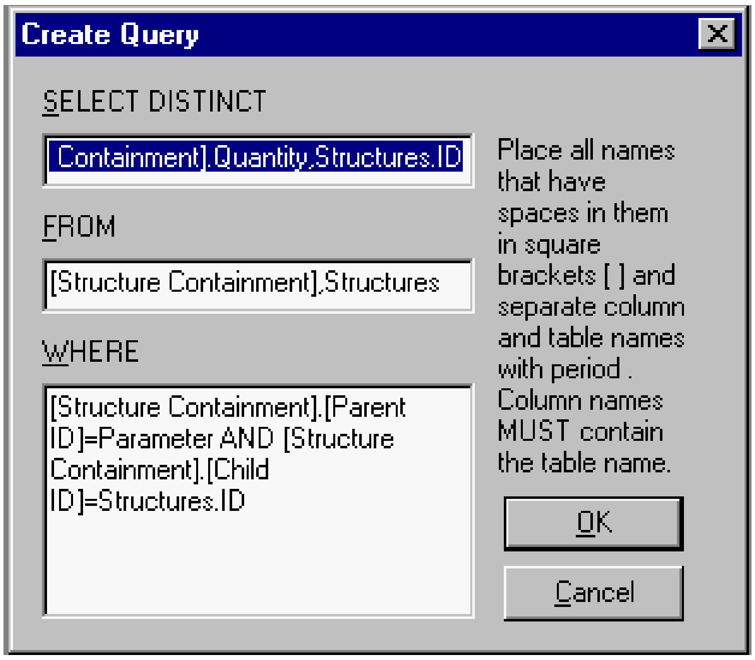


Figure 28: Map Tables Dialog.

The `Map Column` dialog (see Figure 30) allows mapping of any of the column elements. When mapping normal elements, like `title`, the lower part of the dialog is disabled. The lower part is used for `reference` element. The design makes it possible for the database to only holds the file name without path or suffix information. This was crucial to Wärtsilä, because it cannot be expected that the hundreds of subcontractors all have the exact same directory structure. With just a small addition to this dialog (and slightly more logic to the program, of course) it would be possible to show the format of the file since it is not necessary for the file to always be in SGML. What is needed is a new edit box where the notation type of the file is specified. This would generalize the design to work with images, for example.



Create Query

SELECT DISTINCT

Containment].Quantity,Structures.ID

FROM

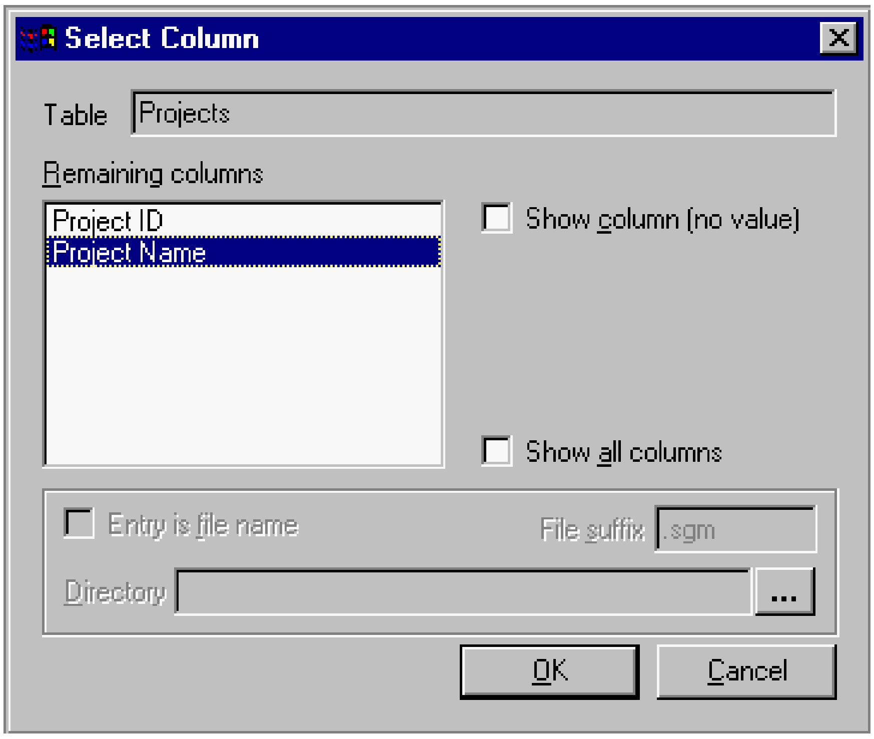
[Structure Containment],Structures

WHERE

[Structure Containment].[Parent ID]=Parameter AND [Structure Containment].[Child ID]=Structures.ID

Place all names that have spaces in them in square brackets [ ] and separate column and table names with period . Column names MUST contain the table name.

Figure 29: Map Queries Dialog.



Select Column

Table: Projects

Remaining columns

- Project ID
- Project Name

☐ Show column (no value)

☐ Show all columns

☐ Entry is file name

File suffix: .sgm

Directory:

Figure 30: Map Columns Dialog.

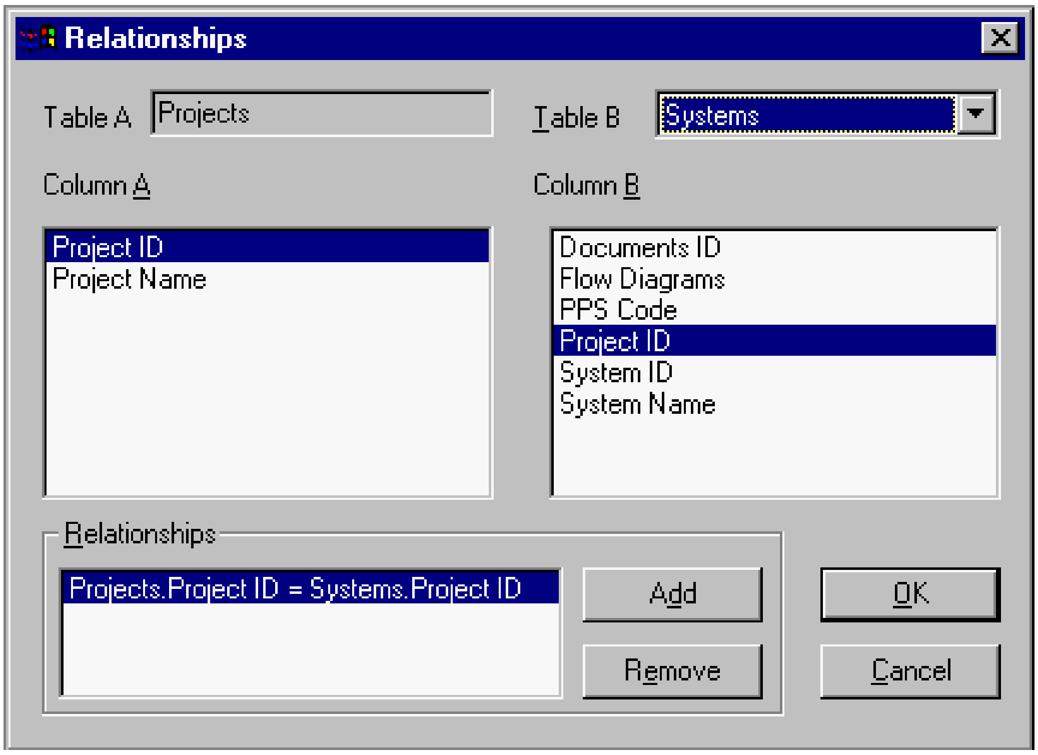


Figure 31: Map Relationships Dialog.

8.2.2 Document Generation

An actual SGML document is generated according to the mappings. At first only the first level `level` elements are generated. As the user navigates the document by clicking on the navigator items (see Figure 23), new queries are sent to the database and the results are inserted in SGML format into the document. The database results are inserted into small SGML template fragments which are then inserted into the document. This approach makes it fast to generate and browse the instance. Generating the full document from even a small one megabyte database would simply take too much time. Figure 32 shows a sample database schema. Figure 33 shows a sample generated instance of it. Appendix C lists the mapping used in its ASCII form.

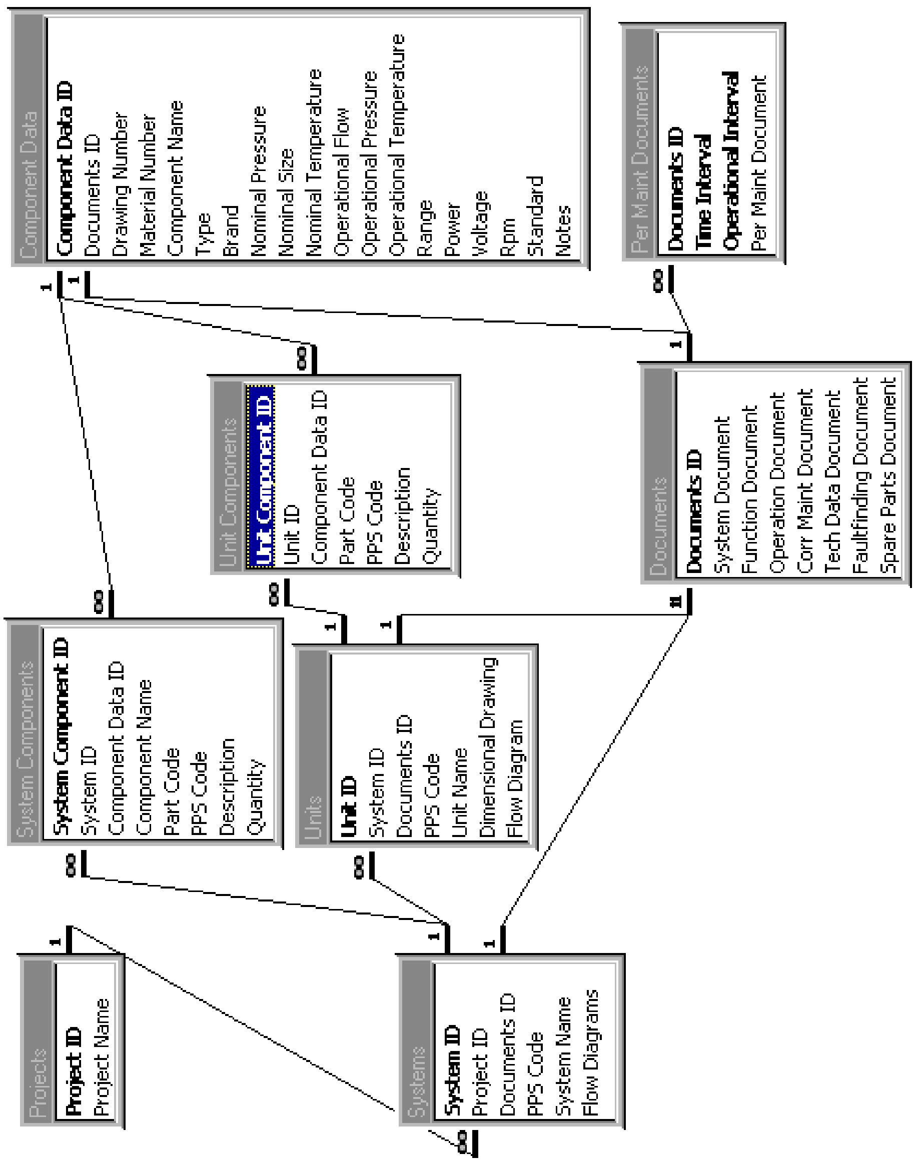
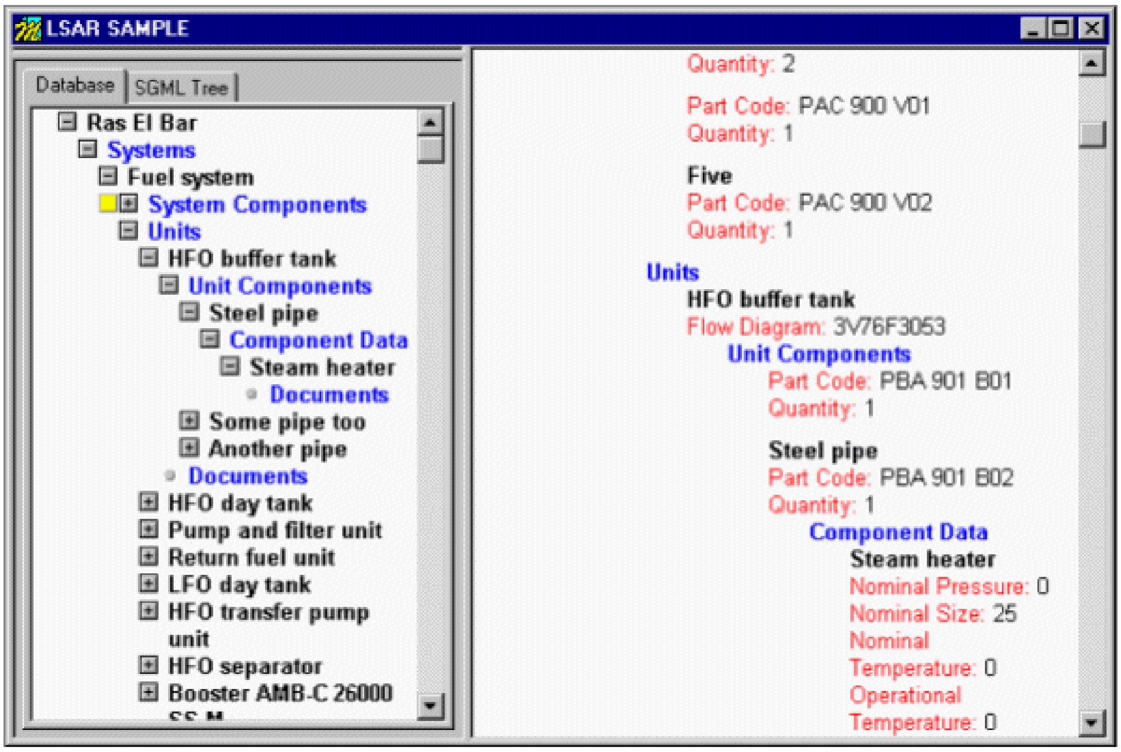


Figure 32: Sample Database Schema.


LSAR SAMPLE

Database | SGML Tree

- Ras El Bar
  - Systems
    - Fuel system
      - System Components
      - Units
        - HFO buffer tank
          - Unit Components
            - Steel pipe
              - Component Data
                - Steam heater
                  - Documents
            - Some pipe too
            - Another pipe
          - Documents
        - HFO day tank
        - Pump and filter unit
        - Return fuel unit
        - LFO day tank
        - HFO transfer pump unit
        - HFO separator
        - Booster AMB-C 26000

Quantity: 2

Part Code: PAC 900 V01
Quantity: 1

Five
Part Code: PAC 900 V02
Quantity: 1

Units
HFO buffer tank
Flow Diagram: 3V76F3053
Unit Components
Part Code: PBA 901 B01
Quantity: 1

Steel pipe
Part Code: PBA 901 B02
Quantity: 1
Component Data
Steam heater
Nominal Pressure: 0
Nominal Size: 25
Nominal Temperature: 0
Operational Temperature: 0

Figure 33: Sample LSAR Generated Document [CIT98].

8.2.3 Publishing

Publishing is done with the Document Set Editor. It is possible to create the whole publication in the editor, but usually an initial set of documents will be created in one of two ways:

- selecting entries from the navigator and opening them in the document set editor, or
- querying the database for documents based on different search criteria.

Multidoc Pro Database Browser and Publisher have a query dialog. It has a simple SQL generator, but it is also possible to write the SQL string by hand. The query will be sent to the query executor. The query executor will notice if the query produces columns that are pointers to file names as specified in the database mapping. If all of the output columns are “document columns“, the query results will be displayed in the document set editor. Otherwise, a tabular view is shown.

After changes are made in the editor, the document can be saved. The editor offers two choices: document set or publication. A document set is mainly intended for electronic browsing. The document set file itself contains links to the actual microdocuments that form the information product. The publication file on the other hand is a large file where all the microdocuments are included. The publication will normally be printed on paper.

8.3 Multidoc Pro Implementation Details

The specifications for the publishing tool were written when the standard Multidoc Pro program was beginning to take shape, but it was not yet released. After the specifications were written, the author was hired by Citec to develop the database and document assembly functionality for Multidoc Pro. At that time a single programmer had been working on the standard Multidoc Pro product for over six months, so the basic architecture was already in place.

The programming language used to develop Multidoc Pro was C++, or, to be exact, Microsoft Visual C++. Multidoc Pro Database Browser and Publisher can only be compiled with version 4.1. The mainline Multidoc Pro code is nowadays compiled with Visual C++ 6.0.

Programming methodology followed mostly [Pro96], although more care was paid to avoid some bad programming practices. For example, many Windows programming guides — and indeed the Visual C++ development environment — seem to use public data members in classes while this is recognized as poor style (for example, [Mey97]).

Typically an MFC (the Microsoft Foundation Class class library that ships with Visual C++) application is based on a document-view architecture, as is Multidoc Pro. A document-view architecture means that there is data (the “document”) that can be projected or shown in many “views”. A change in the document can automatically cause an update in all the views of that document.

Initially there was only one developer for MDP (acronym for Multidoc Pro) and, at the peak of the project, there were six developers working on the same code at the same time. There have been about 10 developers working on the code since the inception of the project. Programmers were assigned to the project for a total of 12 person-years[1]. Of this, about 1.5 years was spent developing the database extensions, during 1996 and 1997, but only six months of this full time.

A version control system was acquired relatively late in the process. Before automated version control snapshots of the code were saved manually every month or so to a Novell server where all developers could access them. Merges were done by hand. This caused errors and delays, so the PVCS Version Manager was finally purchased. Before that Microsoft Visual SourceSafe was tried, but it was deemed too slow and lacking in features.

The whole application was not created from scratch. The Synex ViewPort engine was the core around which the whole application was built and several other packages were acquired to speed the development.

1. The real number of man-years spent on development is significantly less than 12 years because most programmers were also included in other projects.

8.3.1 Synex ViewPort Engine

Synex ViewPort is an SGML engine. It has a fast, non-validating SGML parser. The engine is available on Windows, Unix and Macintosh platforms. On Windows the engine is a Dynamic Load Library (DLL) and has a C Application Programming Interface (API), although the internals are written with C++. There are over 300 API functions. All the ViewPort functions have a prefix `Sv`, for example `SvMoveTagToNext`. All tags, pages and other objects are represented as handles — `HTAG`, `HDOC` and `HPAGE` to name a few.

This section is based mostly on [Syn98], including the images.

Figure 34 shows a simplified view of how data is processed by ViewPort. The formatter and monitor are just names for ViewPort components and have nothing to do with the computer screen. The user application in the project described in this thesis is Multidoc Pro.

Figure 35 shows an overview of the system components of the ViewPort DLL. ViewPort allows customization of most of the components via callback functions. The entity manager was of special interest at one point during the development of the Multidoc Pro database extensions, because it was believed that by customizing the entity manager it would be easy to make ViewPort process database information. Alas, this proved to be a false assumption.

The figure shows how data flows through the various ViewPort components before ending up on the screen. It is possible to customize the behaviour of the components by registering callback functions. For example, it is possible to register a callback for the entity manager. The callback will be called for every entity (see Example 5 for a sample SGML instance with entities) ViewPort detects. The callback function may then process a database query, for example, providing the contents of the entity to ViewPort and signaling that the entity has been handled and that no default behaviour should happen. This approach was tried for Multidoc Pro database extensions. Unfortunately ViewPort needs to resolve all entities when opening a document so this will not work for large databases.

The figure clearly illustrates ViewPort processing and the meaning of various terms used here. It should be very helpful to the reader to study it carefully before moving forward.

ViewPort is at its best in a browser application. Limited support for DTD, editing and validation in general causes troubles when something beyond the functionality of a standard browser is needed. For example, limited support — through an undocumented function — is available for inserting new content into an already open document, but there is no information on whether or not it is possible to delete content.

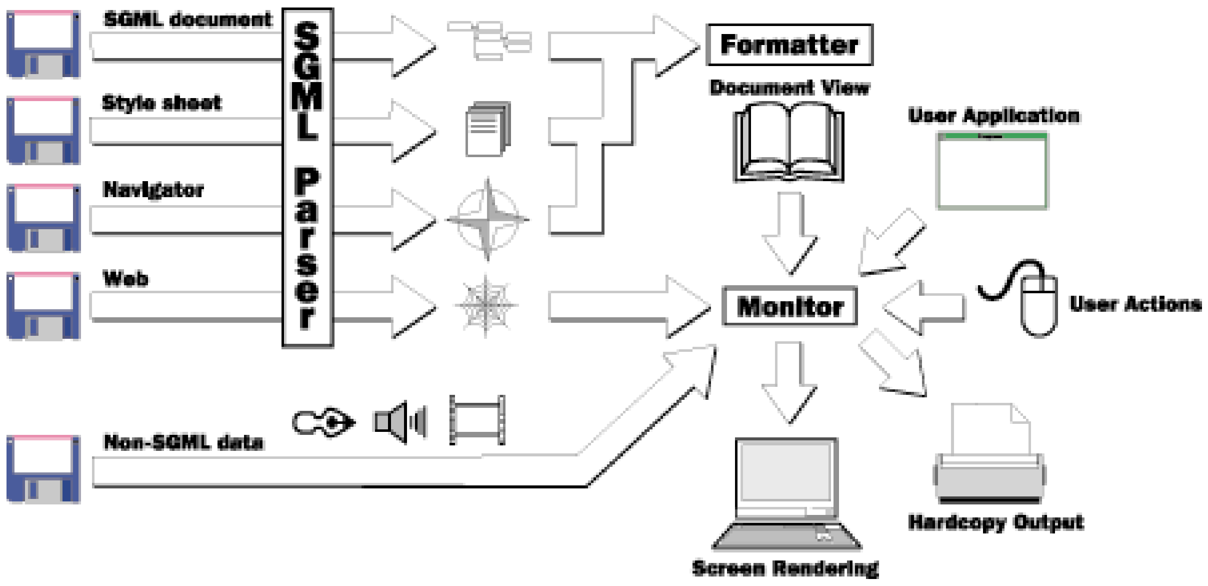


Figure 34: Data Processing in a ViewPort System.

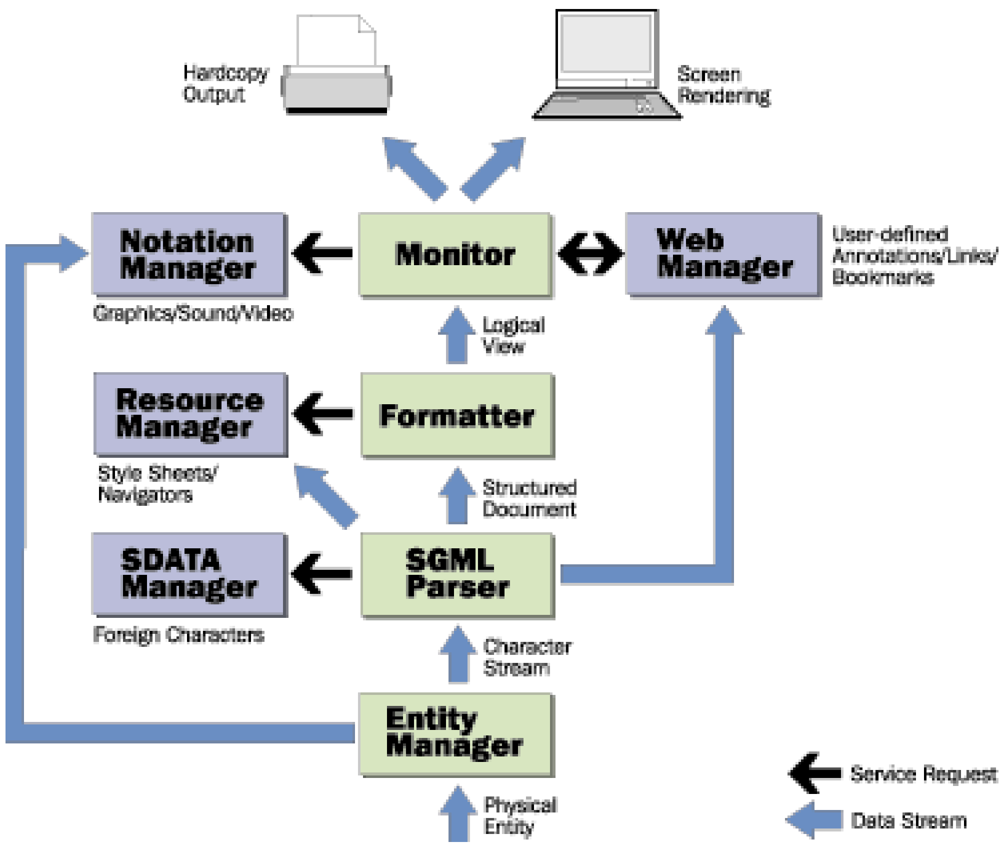


Figure 35: ViewPort System Components.

8.3.2 Other Third-Party Modules

Several smaller software packages were used in the Multidoc Pro products in addition to the Synex ViewPort engine.

The first problem area that was fixed with an additional module was a performance problem with the default tree controls. Multidoc Pro has a special navigator that shows a document's structure as a tree view. With a moderately-sized document opening this tree view took about 20 minutes. The SftTree/DLL was purchased to remedy the situation. It offers extremely fast and customizable tree controls. The tree view that previously took dozens of minutes to open took only a couple of seconds with SftTree/DLL! The SftTree/DLL documentation even claims that creating a tree view with 100,000 entries on a Pentium II 300 MHz takes only about 3 seconds [Sof99]. Interestingly enough, SftTree/DLL is written in C, but it has C and C++ APIs into it. The two C++ frameworks directly supported are MFC and OWL.

Another performance boost came from SmartHeap which replaced the default heap memory handling provided by the Visual C++ compiler. SmartHeap claims to offer from 3 to 100 times faster memory allocation than default compiler-provided allocation [MQ99].

Dialogs in the MFC class library must be specified with fixed-size dimensions. This is very limiting. For example, the old file picker dialog cannot show long filenames. This would not be a problem if the dialog could be stretched so that the filename list box would also grow. This can be accomplished with a neat little class library called NSViews [Nan97] that make the MFC dialogs stretchable. It is naturally possible to specify that certain objects in a dialog cannot be moved or stretched. NSViews is freeware.

The commercial package Objective Toolkit [Rog99], despite its large collection of small utility classes, eventually contributed only a small directory picker. The Objective Plug-in, however, proved invaluable while transforming Multidoc Pro into a Web browser plugin. The Objective Plug-in product seems to be no longer supported.

The database extensions in Multidoc Pro were programmed with the help of Visual SQL from Blue Sky Software[1]. It proved to be invaluable as it it was the only tool we could find that enabled SQL queries to work through ODBC. The enhanced database handling classes were a bit disappointing, but some were used nevertheless. Visual SQL does not seem to be supported anymore. This is most likely because the new database classes in MFC offer everything the Visual SQL classes offered and more.

Multidoc Pro can be downloaded from the web for free evaluation. The evaluation period is 21 days. After that time the program cannot be started because the special CrypKey mechanism prevents this.

Other small ideas, fixes and improvements too numerous to mention were found from the various MFC, C++ and Windows programming resources like mailing lists and web sites.

1. The company homepage is `http://www.blueskysoftware.com/`.

8.3.3 Database Support via Open DataBase Connectivity

The Multidoc Pro Database Browser and Database Publisher were developed for Citec Software Ltd. as a part of this thesis. These products were a part of a larger document management system developed by Citec Engineering Oy for Wärtsilä Diesel (later Wärtsilä NSD). The overall project is described in Chapter 6.

The database connections in the authoring tool and the Multidoc Pro Database Browser and Publisher are handled through Open Database Connectivity (ODBC) [Mic92], a Microsoft standard for connecting to relational databases. ODBC offers a uniform interface for application developers who do not need to worry about the actual database. Most database vendors offer an ODBC interface to their products. There are even ODBC drivers that enable connections over the internet.

The class diagram for the ODBC classes is shown in Figure 36. The `CRecordset` is the standard recordset class in the MFC library, others were coded as part of Multidoc Pro. The `CRecordset` wraps the SQL queries in C++ objects. The `CAbstractRecordset` offers common functionality for the classes derived from it such as properly quoting table and column names. `CColumns` and `CTables` were found from MFC samples and needed very little modifications. Their function is to extract the available columns and tables from the database, respectively.

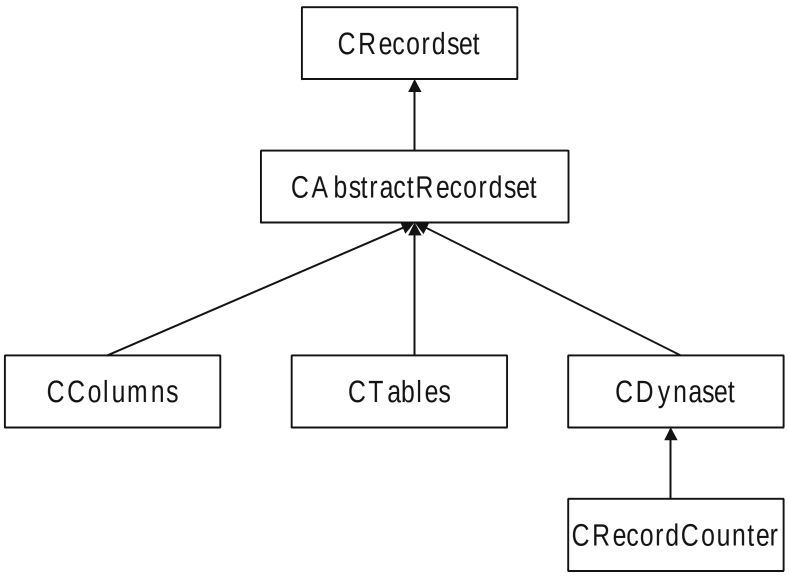


Figure 36: ODBC Recordset Classes.

Because the default `CRecordset` class works only in the situation where the database schema is known in advance, a special `CDynaset` recordset was needed for Multidoc Pro because the database schema could be almost anything. The small utility `CRecordCounter` was based on an example in a newsgroup posting — its function is to count the number of records in a recordset so that progress controls can be used. Database connection was abstracted in the Visual SQL `CVso-`

`Database` class (not shown in the figure), which inherits from the MFC `CDatabase`.

Multidoc Pro generates SQL queries based on user instructions. This generated SQL forms a very small subset of the full SQL available. The grammar for the generated SQL is listed in Table 1. Other types of queries generated transparently by the ODBC API functions may also occur. Queries written by the user may also differ from this grammar.

<query>	SELECT DISTINCT <column list> FROM <table list> [WHERE <condition> [<connective> <condition>]*]
<table list>	<table name>[, <table name>]*
<column list>	COUNT(*) | <table name>.<column name> [,<table name>.<column name>]*
<condition>	<table name>.<column name> <compare> <table name>.<column name> | <field value> | '<field value>'
<table name>	table name | [table name]
<column name>	column name | [column name]
<compare>	= | < | > | <= | >= | <>
<connective>	AND | OR
<field value>	contents of a database field (a row from some column)

Table 1: Multidoc Pro Generated SQL Grammar.

8.3.4 Main Functionality Classes

The majority of the database-specific code is in the class `HTDBSGMLDoc`. What first started out as an attempt to “make objects responsible for their own user interfaces” (Allen Holub), quickly became the **blob** antipattern [Bro98]. The blob antipattern refers to a design flaw in which a huge class is created that has most of the functionality in the system. It should be fixed by dividing the work more evenly between different classes.

The code uses beneficial design patterns as well. For example, the `HTDTD` object that contains some knowledge about the hardcoded DTD is a **singleton** ([Gam94], [Mey97] and more in [Vli98]).

The `HTDBSGMLDoc` talks to the database, updates the controls in all the database-specific dialogs, builds the database connected SGML document, creates the publications and so on. It uses some helpers to manage all this, acting as a **bridge** design pattern for some of its components.

Figure 37 shows the classes that incorporate the main functionality in the database extensions. `HTCollection` is a template for collections. `HTDBSGMLDocCollection` and `HTDTDTreeItemDataCollection` instantiate concrete versions of the template. `HTDTDTreeItemData` holds the information about the database mapping nodes. This information is used when generating the SGML documents from databases. The black diamonds with lines indicate the "has-a" relationship, i.e., the class with the diamond has the other class as a data member. This is somewhat simplified Unified Modelling Language (UML) [Boo98] notation.

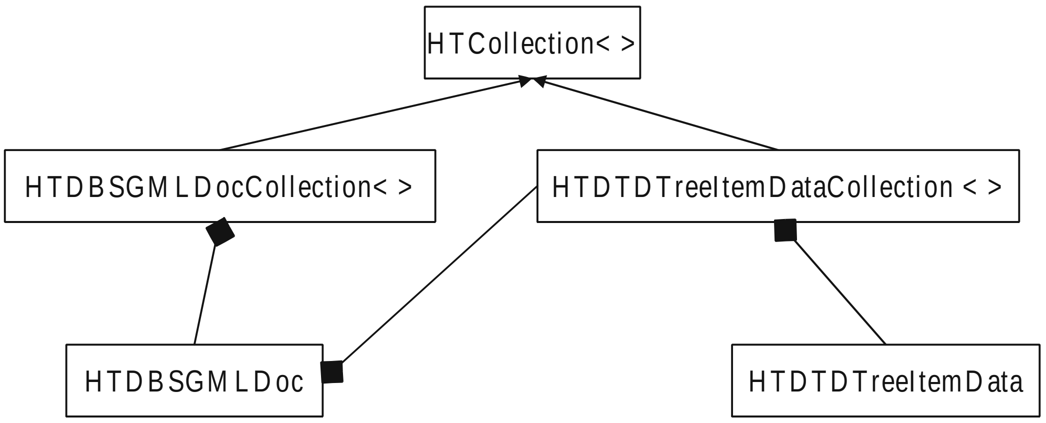


Figure 37: Main Database Extensions Classes.

8.3.5 Code Metrics

Table 2 lists some code metrics about Multidoc Pro. The full Multidoc Pro includes code for standard Multidoc Pro and all the variations of it, including Translating Editor and customized browsers. Binary files (icons, bitmaps, etc.) and third party code is excluded. Metrics were extracted with [Van98] so they are not as accurate as they could be. Nevertheless, the figures show that the database extensions are roughly 20% of the full Multidoc Pro source code.

	Full Multidoc Pro	Of Which Database Extensions[a]
File Count	252	77
Text Lines	92736	17862
Semicolons	34076	7313
Comments	12249	1835
%Semicolons	36	40
%Comments	13	10
Classes	190	40
Data Members	1553	337
Member Functions	2625	552

a. Approximate figures, code mixed within files of standard Multidoc Pro not counted.

Table 2: Multidoc Pro Code Metrics.

8.3.6 Testing

Testing of Multidoc Pro was difficult at times due to the integration of the various third party components, some of which were only in binary format. If an error was tracked down to a third party module, it was usually necessary to submit a bug report to the manufacturer and to pray that they would fix it in a timely fashion. If a fix was not promised, or was taking too long, workarounds had to be figured out. This was also a difficult task because of the black box nature of some components.

Numega BoundsChecker[1] software was purchased mainly to track memory related bugs. BoundsChecker required recompilation of the source to instrument the code. The BoundsChecker compile also found some bugs. During run-time BoundsChecker can cause a pop-up a message box to appear informing the user of memory leak or other kinds of errors such as uninitialized variables. Errors can be filtered and logged as well. BoundsChecker offers different levels of instrumentation, each level catching more errors. Unfortunately, the most rigorous level of compilation was unusable with Multidpc Pro as it always got into an endless loop.

1. BoundsChecker is a product of Compuware Corporation. See information about BoundsChecker from URL `http://www.numega.com/products/aed/vc.shtml`.

The database was another great source of grief, at least in the beginning. When SQL queries were received from the database designer and tried with the ODBC classes, they invariably failed with Microsoft Access. The ODBC log and the error messages were of some help in tracking down the causes of failures. The biggest help was the Visual SQL package. Rewrite of the query with the Visual SQL query editor/generator always gave a suitable query for the ODBC classes.

In addition to the testing performed at development time, Multidoc Pro Database Publisher and Browser were tested in real use situations both at Citec and Wärtsilä. Several of these kinds of test versions of Multidoc Pro were prepared during the development to gather feedback and find bugs.

8.4 In Retrospect

Multidoc Pro Database Browser and Publisher have showed a way to view relational databases as SGML data. Although the idea of a relational database keeping track of SGML information is not new, the combination of the Author Tool and the Multidoc Pro Database tools have made the PIM system relatively efficient and easy to use.

The implemented system accomplished what it was supposed to do: assemble large documents from document fragments managed by databases. Many things can be automated, so customized information production is easy. Needless to say, there is still room for improvement.

The implementation of Multidoc Pro Database Browser and Publisher taught us some lessons. It was learned that ViewPort was not too well-suited for the job. More control over the DTD and some of functionality that is used in an editor was sorely needed. Because ViewPort was basically intended for browsers, it does not handle insertion and deletion of content very well.

The save file format should have been SGML in all cases. This could actually be taken a step further by keeping some of the information in memory resident SGML files instead of normal C/C++ structures.

There were two major problems in the project. Probably the bigger one was that it was not always clear what was needed and how the full system would work and be integrated together. And even if the intent was clear in the beginning, the objectives were redefined during the project. As usual, more planning would have saved some work in the later stages. All this caused unnecessary work and concentration on things that were not important. Another problem was ViewPort in the implementation of the database extensions. There were many cases were ViewPort simply was not flexible enough and an alternative had to be found. In one case, there was even an undocumented function that would have solved a problem but it took about six months from the initial question to the ViewPort manufacturer for them to reveal that they indeed already had a solution!

It is not too difficult to see what changes would be beneficial for the Multidoc Pro Database Browser and Publisher if they are ever developed further. The mapping does not allow mapping arbitrary structures from database. This is the most urgently needed improvement. Another change in this direction would be to allow arbitrary database DTDs. A better user interface would make the database mapping easier to accomplish. Publication is, in fact, the most important feature of the Database Publisher and warrants special attention. Speed is also an issue that should be looked into. Some things that would also need more work, but do not concern the program as such, are better help and better sample databases and mappings. Finally, there are some annoying bugs in the document assembly procedures that sometimes cause micro-documents to fail to nest properly in the document set or publication.

The other Multidoc Pro products have received good reviews. The product family has grown to include translation tools and the like, and this progress is likely to continue. However, the Authoring Tool is as good as dead and buried, and the Multidoc Pro Database Browser and Publisher fare not much better. The database extensions were designed and implemented to be more generic tools, which is why some copies of it has been sold to other customers as well. Unfortunately, not enough to justify further development at this time.

Although the SGML engine ViewPort has been a good choice, it is based on old ideas. With the new HyTime standard the whole parser should be based on the grove idea. The ViewPort-specific formatting language must be replaced with a standard solution like DSSSL. What all this means is that a new version of ViewPort must be based on these standardized ideas or a completely new SGML/HyTime engine is needed. In fact, that is where the development seems to be going.

In March 1998 Netscape released the source to its web browser. Citec saw this as a great opportunity, and has been working with the code since then. Citec has already gained reputation dealing with the huge and difficult piece of code that will be the future Netscape Communicator 5.0. Citec's own SGML/XML-enhanced browser DocZilla is based on the same source — with HyTime linking support. And who knows, maybe the database support will be incorporated into DocZilla at some point.

Chapter 9

Summary

Only fools prefer the past.

Frank Herbert

Using relational databases to manage product documentation is not a new concept. There are technically better alternatives, but technology alone rarely drives business. Relational databases dominate the market and many organizations are already using relational databases. To make them into document management system could be as simple as adding a single table to the product information database. This new table would holds references to product documents which are saved on a normal file server. Of course, new relationships must be added, but that is about all that is required of the database.

The weakness of this model is that it is very easy to break the reference from the database to the file system — for example, by simply renaming files on the file system.

More work is required with the tools that interact with the product information database. There must be a special authoring tool that makes it easy to write product documentation and tie it in with the records in the database. Additionally, a publishing tool is needed that can create manuals from the document fragments managed by the database.

The most difficult challenge, however, is managing people. After all is said and done, it is of no use if the people in question do not accept the change or if they are not given enough training and time to move to the new system.

Even though the project with Wärtsilä did not go as was planned, the implementation of the system showed it is not too difficult, technically, to implement a document management system the way it is described in this paper. Although many publications mention this technique of using relational databases to manage SGML documents or document fragments, they rarely disclose the details and difficulties involved with it. The participants in this project learned almost everything the hard way. A second try, if there ever will be one, should be almost a guaranteed success.

References

[Ang97] Angerstein Paula, **Why Your Document Management System Should Care About Hyperlinks**, available on the WWW at <URL: `http://www.texcel.no/se97talk.htm`>, Texcel Research, Inc., 1997.

[Arb95] ArborText, Inc., **Getting Started with SGML: A Guide to the Standard Generalized Markup Language and Its Role in Information Management**, available on the WWW at <URL: `http://www.arbortext.com/wp.html`>, 1995.

[Bal97] Balasubramanian V., Bashian Alf, Porcher Daniel, **A Large-Scale Hypermedia Application Using Document Management And Web Technologies**, in "HYPERTEXT '97", Proceedings of the Eight ACM Concerence on Hypertext, pages 134-145, 1997.

[Boo98] Booch Grady, Jacobson Ivar, Rumbaugh James, **The Unified Modeling Language User Guide**, Addison-Wesley Publishing Company, 1998.

[Bou99] Bourret Ronald, **XML and Databases**, available on the WWW at <URL: `http://www.informatik.tu-darmstadt.de/DVS1/staff/bourret/xml/XMLAndDatabases.htm`>, Technical University of Darmstadt, 1999.

[Bro98] Brown William J., Malveau Raphael C., Brown William H., McCormick III Hays W., **Antipatterns: Refactoring Software, Architecture and Projects in Crisis**, John Wiley & Sons, 1998.

[Bus45] Bush Vannevar, **As We May Think**, The Atlantic Monthly, July (1945), pages 641-649.

[Böh94] Böhm Klemens, Aberer Karl, **Storing HyTime Documents In an Object-Oriented Database**, in: "CIKM '94", Proceedings of the Third International Conference on Information and Knowledge Management, pages 26-33, 1994.

[CIM98] CIMdata, **Product Data Management: The Definition: An introduction to Concepts, Benefits, and Terminology**, available on the WWW at <URL: `http://www.cimdata.com`>, 1988.

[CIT97a] CITEC Engineering Oy, **Multidoc Pro Database Browser and Database Publisher — User's Manual**, 1997.

[CIT97b] CITEC Engineering Oy, **WNS Author Tool for the Base-DTD User Manual**, 1997.

[CIT98] CITEC Engineering Oy, **Multidoc Pro Browser/Publisher Product Brief**, 1998.

[DeR94] DeRose Steven J., Durand David G., **Making Hypermedia Work: A User's Guide to HyTime**, Kluwer Academic Publishers, 1994.

[Eck95] Eckel Bruce, **Thinking in C++**, Prentice Hall, Inc., 1995.

[Elo95] Elovainio Kimmo, **SGML-Based Documentation Process**, VTT OFFSETPAINO, 1995.

[FMV95] FMV, **Description of FMV Grund-DTD**, available on the WWW at <URL: `http://info.admin.kth.se/SGML/Bibliotek/DTDer/FMVGrund-DTD/`>, 1995.

[Gam94] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison–Wesley, 1994.

[Gol90] Goldfarb Charles F., **The SGML Handbook**, Oxford University Press Inc., 1990.

[Hof99] Hoffman James, **Introduction to Structured Query Language**, available on the WWW at <URL: `http://w3.one.net/~jhoffman/sqltut.htm`>, 1999.

[ISO86] ISO 8879:1986, **Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML)**, 1986.

[ISO89] ISO 8613, **Information Technology — Text and Office Systems — Office Document Architecture (ODA)**, 1989.

[ISO92a] ISO/IEC 9075:1992, **Information Technology — Database Languages — SQL**, 1992.

[ISO92b] ISO/IEC 8632:1992, **Information Processing Systems — Computer Graphics Metafile for the Storage and Transfer of Picture Description Information (CGM)**, 1992.

[ISO93] ISO/IEC 10646-1:1993, **Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane**, 1993.

[ISO94] ISO 10303, **Industrial Automation Systems and Integration — Product Data Representation and Exchange (STEP)**, 1994-1998.

[ISO96] ISO/IEC 10179:1996, **Information Technology — Text and Office Systems — Document Style Semantics and Specification Language (DSSSL)**, 1996.

[ISO97] ISO/IEC 10744:1997, **Information Technology — Hypermedia/Time-based Structuring Language (HyTime)**, 1997.

[ISO98a] ISO/IEC 14772-1:1998, **Information technology — Computer graphics and image processing — The Virtual Reality Modeling Language — Part 1: Functional specification and UTF-8 encoding (VRML)**, 1998.

[ISO98b] ISO/IEC 16262:1998, **Information Technology — ECMAScript Language Specification**, 1998.

[Kim97] Kimber W. Eliot, **A Tutorial Introduction to SGML Architectures**, available on the WWW at <URL: `http://www.isogen.com/papers/archintro.html`>, ISOGEN International Corp., 1997.

[Kim98] Kimber W. Eliot, **Practical Hypermedia: An Introduction to HyTime**, Prentice Hall, 1998.

[Kla98] Klavans Judith, **Data Bases in Digital Libraries: Where Computer Science and Information Management Meet**, in "PODS '98", proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems, pages 224-226, 1998.

[Lam94] Lamport Leslie, **LaTeX: A Document Preparation System**, Addison-Wesley Publishing Company, Inc, 1994.

[Lin97] Lindén Greger, **Structured Document Transformations**, PhD Thesis, Series of Publications A, University of Helsinki, Report A-1997-2 (1997).

[Loi99] Loizou George, Levene Mark, **A Guided Tour of Relational Databases and Beyond**, Springer Verlag, 1999.

[Loo98] Loomis Mary, Chaudri Akmal B., **Object Databases in Practice**, Prentice-Hall, Inc, 1998.

[Mal95] Maler Eve, Andaloussi Jeanne El, **Developing SGML DTDs: From Text to Model to Markup**, Prentice Hall, 1995.

[Met99] Metsäranta Pekka, **"Rakenteisen tiedon säilyttäminen: XML-dokumentti OAIS-viitemallissa (in Finnish)"**, Master of Science Thesis, Jyväskylä University, available on the WWW at <URL: `http://www.syspro.fi/pekka.metsaranta/gradu/`>, 1999.

[Mey97] Meyers Scott, **Effective C++: 50 Ways to Improve Your Programs and Designs**, Addison-Wesley Publishing Company, 1997.

[Mic92] Microsoft Corporation, **ODBC Application Programmer's Guide**, Microsoft, 1992.

[MQ99] MicroQuill, **SmartHeap**, available on the WWW at <URL: `http://www.microquill.com/prod_sh/index_sh.htm`>, 1999.

[Mya98] Myaeng Sung Hyon, Jang Don-Hyun, Kim Mun-Seok, Zhoo Zong-Cheol, **A Flexible Model for Retrieval of SGML Documents**, in "SIGIR '98", Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 138-145, 1998.

[Nan97] NanoSoft Corporation, **NSViews Version 1.04**, available on the WWW at <URL: `http://www.nanocorp.com/nsviews/default.htm`>, 1997.

[Nel82] Nelson Theodore Holm, **Literary Machines**, Mindful Press, 1982.

[Nel97] Nelson Theodore Holm, **Embedded Markup Considered Harmful**, available on the WWW at <URL: `http://www.xml.com`>, 1997.

[New91] Newcomb Steven R., Kipp Neill A., Newcomb Victoria T., **“HyTime”: The Hypermedia/Time-based Document Structuring Language**, Communications of the ACM, Vol. 43, No. II (1991).

[OAS99] OASIS, **The DocBook DTD**, available on the WWW at <URL: `http://www.oasis-open.org/docbook/`>, 1999.

[Onn99] Onnela Tapio, **Bittiarkisto voi jäädä lukematta (in Finnish)**, Tiede 2000, 5 (1999), p. 37.

[Paq92] Paquet Gaël, **Hyper9002: An Online Operating Manual for a Chemical Manufacturer Using Hypertext Integrated with an Object Oriented Database**, in: “SAC ‘92”, Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing (vol. II): Technolological Challenges of the 1990’s, pages 976-984, 1992.

[PDM97a] The PDM Information Center, courtesy of Hewlett-Packard, **Understanding Product Data Management**, available on the WWW at <URL: `http://www.pdmic.com/undrstnd.html`>, 1997.

[PDM97b] The PDM Information Center, **How the technology Has Evolved: A Short Review**, available on the WWW at <URL: `http://www.pdmic.com/evolt-ech.html`>, 1997.

[Pel97a] Peltonen Björn, Mäki Erik, **Case Study: Wärtsilä Diesel Oy, Power Plants**, available on the WWW at <URL: `http://www.citec.fi/company/services/case/wd_pp.html`>, 1997.

[Pel97b] Peltonen Björn, **“Case Study”, The SGML (Standard Generalized Markup Language) Implementation at Norsk Hydro: Do More with Less and Do It Better**, in: “SGML Finland 1997 — seminaarijulkaisu”, Proceedings of Finnish SGML Conference, SGML User’s Group Finland, pages 4-9, 1997.

[Pre98] Prescod Paul, **Formalizing SGML and XML Instances and Schemata with Forest Automata Theory**, available on the WWW at <URL: `http://www.prescod.net/forest/shorttut/`>, 1998.

[Pro96] Prosise Jeff, **Programming Windows 95 with MFC: Create Programs for Windows Quickly with the Microsoft Foundation Class Library**, Microsoft Press, 1996.

[Rei98] Reinwald Berthold, Pirahesh Hamid, **SQL Open Heterogenous Data Access**, in "SIGMOD '98", Proceedings of ACM SIGMOD International Conference on Management of Data, pages 506-507, ACM, 1998.

[Rog99] RogueWave Software, **Objective Toolkit**, available on the WWW at <URL: `http://www.roguewave.com/products/ot/`>, 1999.

[Ryt97] Rytkönen Kimmo, Kunz Jürgen, **DOCSTEP — Technical Documentation Creation and Management using STEP**, in: "SGML Finland 1997 — seminaarijulkaisu", Proceedings of Finnish SGML Conference, SGML Finland User's Group, pages 39-68, 1997.

[Sip96] Sipser Michael, **Introduction to the Theory of Computation**, International Thomson Publishing, 1996.

[Sof99] Softel vdm Inc., **SftTree/DLL 4.0 Product Information**, available on the WWW at <URL: `http://www.softelvdm.com/sfttree.html`>, 1999.

[Som96] Sommerville Ian, **Software Engineering**, Addison–Wesley Publishers Ltd., 1996.

[Sun81] Sundgren Bo, **Databaser och datamodeller (in Swedish)**, Studentlitteratur, 1981.

[Syn98] Synex Information AB, **Synex ViewPort Version 2.1 Programmer's Manual**, 1998.

[Tra95] Travis Brian, Waldt Dale, **The SGML Implementation Guide: A Blueprint for SGML Migration**, Springer-Verlag, 1995.

[Tur96] Turner Ronald C., Douglass Timothy A., Turner Audrey J., **README.1ST: SGML For Writers and Editors**, Prentice-Hall, Inc, 1996.

[Van98] Vanvliet Peter A., **CodeCount**, available on the WWW at <URL: `http://www.nanocorp.com/vanvliet/peter/codecount/codecount.htm`>, 1998.

[Vli98] Vlissides John, **Pattern Hatching: Design Patterns Applied**, Addison-Wesley Publishing Company, 1998.

[W3C96] World Wide Web Consortium, **Cascading Style Sheets (CSS)**, available on the WWW at <URL: `http://www.w3.org/TR/REC-css`>, 1996.

[W3C98] World Wide Web Consortium, **Extensible Markup Language (XML) 1.0**, available on the WWW at <URL: `http://www.w3.org/TR/REC-xml`>, 1998.

[W3C99a] World Wide Web Consortium, **Namespaces in XML**, available on the WWW at <URL: `http://www.w3.org/TR/REC-xml-names/`>, 1999.

[W3C99b] World Wide Web Consortium, **Extensible Stylesheet Language (XSL) Specification: W3C Working Draft 21 April 1999**, available on the WWW at <URL: `http://www.w3.org/TR/WD-xsl/`>, 1999.

[Wak99] Wakizono Ryuji, Kawamura Toshikazu, Tsuchiya Takehiko, Hatanaka Takahiro, Tanaka Tatsuji, **Object-Oriented Database Management System for Process Control Systems -Development and Evaluation-**, in "SAC '99", Proceedings of the 1999 ACM Symposium on Applied Computing, pages 204-209, ACM, 1999.

[Whi99] Whitehorn Mark, Marklyn Bill, **Inside Relational Databases: With Examples in Access**, Springer-Verlag, 1999.

[Yar99] Yarger Randy Jay, Reese George, King Tim, **MySQL & mSQL**, O'Reilly & Associates, Inc., 1999.

Appendix A

Database DTD

This appendix presents the predefined, hardcoded database DTD used by Multidoc Pro Database Browser and Publisher. It should be noted that the actual DTD is slighty more detailed than what the user sees (and what is coded in the program). The reason for this is that the actual DTD anticipates some enhancements and changes in the future.

```
<!--******************************************************************-- -
-* *--
--* Database DTD for MultiDoc PRO *--
--* Version 1.0 *--
--* May 5, 1997 *--
--* *--
--* *--
--* Joakim Östman, Oy CITEC AB Information Technology *--
--* Minor modifications by Heikki Toivonen, CITEC *--
--* *--
--* *--
--* Typical doctype declaration of document sets *--
--* <!DOCTYPE Database PUBLIC *--
--* "-//CITEC INFORMATION TECHNOLOGY//DTD Database//EN" "DATABASE.ENT" *--
--******************************************************************-->
<!DOCTYPE database [
<!ELEMENT database - O (level+) >
<!ATTLIST database
database CDATA #IMPLIED >

<!ELEMENT level - O (title? , (data* , level*)) >
<!ATTLIST level
table CDATA #IMPLIED
subdoc CDATA #IMPLIED >

<!ELEMENT title - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST title
column CDATA #IMPLIED >

<!ELEMENT superscript - O (#PCDATA) >

<!ELEMENT subscript - O (#PCDATA) >

<!ELEMENT data - O (dataname? , datavalue*) >

<!ELEMENT dataname - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST dataname
column CDATA #IMPLIED >

<!ELEMENT datavalue - O (datadescription* | reference* | (value , unit?)*)
>

<!ELEMENT datadescription - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST datadescription
column CDATA #IMPLIED >

<!ELEMENT reference - O (ref.name? , nameloc) >
<!ATTLIST reference
id ID #REQUIRED
mediatype (SGML , NON-SGML) "SGML"
linkend IDREF #REQUIRED
HyTime NAME "clink"
column CDATA #IMPLIED >

<!ELEMENT ref.name - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST ref.name
column CDATA #IMPLIED >

<!ELEMENT nameloc - O (nmlist) >
<!ATTLIST nameloc
id ID #REQUIRED
HyTime NAME "nameloc" >

<!ELEMENT nmlist - O (#PCDATA) >
<!ATTLIST nmlist
docorsub ENTITY #IMPLIED
nametype (element , entity) "element"
HyTime NAME "nmlist" >

<!ELEMENT value - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST value
column CDATA #IMPLIED >

<!ELEMENT unit - O ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST unit
column CDATA #IMPLIED >
]>
```

Appendix B

Database DTD 2

This appendix shows an attempt to fix some of the shortcomings of the first version of the database DTD that was implemented in Multidoc Pro Database Browser and Publisher. This DTD can store all the information about the connection to the database and the queries needed to dynamically build the SGML document while browsing the document instance. No hidden data would be needed. Queries can be reused for greater efficiency.

The document language specified in this DTD is both the database mapping save format and the format of the dynamically built SGML document.

This DTD is a work in progress and has not been tested to offer everything that would be needed.

```
<!--*****************************************************************--
--* *--
--* Database DTD for MultiDoc PRO *--
--* Version 1.5 *--
--* Jan 31, 1998 *--
--* *--
--* (c) CITEC Engineering Oy 1996-1998 *--
--* Heikki Toivonen *--
--* *--
--* *--
--* Typical doctype declaration *--
--* <!DOCTYPE Database PUBLIC *--
--* "-//CITEC INFORMATION TECHNOLOGY//DTD Database//EN" "DATABASE.ENT" *--
--*****************************************************************-->
<!-- TODO: Parameter passing/relationships/opening levels-->
<!-- General notice about attribute values: - quotes are double quotes (")
except when the value contains double quotes - if the value contains
double quotes, the quotes will be single quotes (’) - if the value
contains both single and double quotes, the quotes will be double quotes
and double quotes in the value will be replaced with an "entity" (&gt;)
-->
<!-- If this DTD is used to save a database mapping configuration, the
root element will be configuration. For documents generated based on the
configuration the root element is either databases or database depending
on the windows attribute. -->

<!ELEMENT configuration - - (configuration-title? , databases) >
<!ATTLIST configuration
id ID #REQUIRED
```

```
windows (multi | single) "multi" -- multi opens each database element --
-- in its own window --
>

<!ELEMENT configuration-title - - (#PCDATA) >

<!ELEMENT databases - - (databases-title? , database+) >
<!ATTLIST databases
id ID #REQUIRED >

<!ELEMENT databases-title - - (#PCDATA) >

<!ELEMENT database - - (dsn, database-title? , meta , level+) >
<!ATTLIST database
id ID #REQUIRED
generated-levels NUMBER "1" -- Initial number of levels to generate, --
-- negative value means infinite levels --
database CDATA #REQUIRED -- ODBC Data Source Name --
>

<!ELEMENT database-title - - (#PCDATA) >

<!ELEMENT meta - - (stylesheets?, navigators?, webs?, queries? , contents?
, URL-directories? , suffixes? , ndatas?) >

<!ELEMENT stylesheets - - (stylesheet+) >

<!ELEMENT stylesheet - - (name,file) >

<!ELEMENT name - - (#PCDATA) >

<!ELEMENT file - - (#PCDATA) >

<!ELEMENT navigators - - (navigator+) >

<!ELEMENT navigator - - (name,file) >

<!ELEMENT webs - - (web+) >

<!ELEMENT web - - (name,file) >
<!ATTLIST web
web (web | docweb) "web" -- CHECK THIS! --
>

<!-- Queries are stored here so they can be reused and it is easier to
change them. The queries are referenced from the elements using them. -->
<!ELEMENT queries - - (query+) >

<!ELEMENT query - - (sql) >
<!ATTLIST query
id ID #REQUIRED >

<!-- The SQL text of the query could be represented without SGML structure
if there was a suitable SQL parser. Alternative/addition is to have a DTD
```

fragment describing the structure of SQL queries. Below is a simple version.
Example 1: SELECT DISTINCT Doctors.Name,Rooms.* FROM Doctors,Rooms WHERE (Doctors.Age > 50) AND ((Rooms.Wing = ’East’) AND NOT (Doctors.Patients >= 100)) ;

```
<sql id="sql-1">
<clause>
<select distinct="distinct">
<columns>
<table-column-entry>
<table>Doctors</table>
<column>Name</column>
</table-column-entry>
<table-column-entry>
<table>Rooms</table>
<column all="all"></column>
</table-column-entry>
</columns>
</select>
<from>
<tables>
<table>Doctors</table>
<table>Rooms</table>
</tables>
</from>
<where>
<where-group>
<table-column-entry>
<table>Doctors</table>
<column>Name</column>
</table-column-entry>
<operator operator="lt">
<column-value>50</column-value>
</where-group> <
where-group logical-operator="AND">
<where-group>
<table-column-entry>
<table>Rooms</table>
<column>Wing</column>
</table-column-entry>
<operator operator="eq">
<column-value quotes="quotes">East</column-value>
</where-group>
<where-group logical-operator="AND" reverse="reverse">
<table-column-entry>
<table>Doctors</table>
<column>Patients</column>
</table-column-entry>
<operator operator="ge">
<column-value>100</column-value>
</where-group>
</where-group>
</where>
```

```
</clause>
</sql> -->

<!ELEMENT sql - - (clause+ , order-by?) >
<!ATTLIST sql
id ID #REQUIRED >

<!ELEMENT clause - - (clause* | (select , from , where? , order-by?) ) >
<!ATTLIST clause
union CDATA #FIXED "UNION" -- ignored for 1st clause -- >

<!ELEMENT select - - (columns*) >
<!ATTLIST select
distinct (distinct | nodistinct) "distinct"
all (all | notall) "notall" -- if all, semantic error to have columns --
count (count | nocount) "nocount" -- if count, semantic error to have
columns -- >

<!ELEMENT columns - - (table-column-entry+) >

<!ELEMENT table-column-entry - - (table , column) >

<!ELEMENT table - - (#PCDATA) >
<!ATTLIST table
all (all | notall) "notall" -- if all, semantic error to have columns all
--
-- also semantic error to have content -- >

<!ELEMENT column - - (#PCDATA) >
<!ATTLIST column
all (all | notall) "notall" -- if all, semantic error to have tables
all --
-- also semantic error to have content -- >

<!ELEMENT from - - (tables+) >

<!ELEMENT tables - - (table+) >

<!ELEMENT where - - (where-group+) >

<!ELEMENT where-group - - ( where-group* | (table-column-entry , operator
, column-value) ) >
<!ATTLIST where-group
logical-operator (AND | OR) #IMPLIED -- semantic error on first, otherwise
required --
reverse (reverse | normal) "normal" -- NOT -- >

<!ELEMENT operator - o EMPTY >
<!ATTLIST operator
operator (eq | ne | lt | le | gt | ge | like) #REQUIRED
-- = <> < <= > >= LIKE -- >

<!ELEMENT column-value - - (#PCDATA | table-column-entry) >
<!ATTLIST column-value
```

```
quotes (quotes | noquotes) "noquotes" >

<!ELEMENT order-by - - (columns+) >
<!ATTLIST order-by direction (asc | desc) "asc" >

<!-- Contents, URL-directories, suffixes and NDATAs are also reusable.-->
<!ELEMENT contents - - (content+) >

<!ELEMENT content - - (#PCDATA) >
<!ATTLIST content
id ID #REQUIRED >

<!ELEMENT URL-directories - - (dir+) >

<!ELEMENT dir - - (#PCDATA) >
<!ATTLIST dir
id ID #REQUIRED >

<!ELEMENT suffixes - - (suffix+) >

<!ELEMENT suffix - - (#PCDATA) >
<!ATTLIST suffix
id ID #REQUIRED >

<!ELEMENT ndatas - - (ndata+) >

<!ELEMENT ndata - - (#PCDATA) >
<!ATTLIST ndata
id ID #REQUIRED >

<!-- A level element has a pointer to its query. Some of the query’s
output columns are mapped to its child elements like title,
datadescription and so on. NOTE: It would be possible to required explicit
output columns. Then it would be possible to point from a column-mappable
element to a table-column-entry. A level element can have multiple
relationships with its children. At least one relationship must exist
between a parent element and each of its children. During the mapping the
left and right-hand side table-column-entry’s are filled. When a root
level (no ancestor level elements) element is being generated, its
relationships mappings are checked, and the level element’s query is
modified accordingly. After the query has been resolved and the level SGML
is being written, the column-value parts of the relationships will be
written. When a child level item is being generated, the parent levels
relationships are checked, as well as the current relationships, and the
actual query is modified accordingly. The child can find the correct
relationships with pointers to it’s parent’s relationships. -->
<!ELEMENT level - - (title? , level-meta? ,(data* , level*)) >
<!ATTLIST level
query IDREF #REQUIRED
relationships CDATA #IMPLIED -- parent level’s relationship ids -- >

<!ELEMENT level-meta - - (relationship+) >

<!ELEMENT relationship - - (table-column-entry , table-column-entry ,
column-value?) >
<!ATTLIST relationship
id CDATA #REQUIRED -- CDATA because multiple same id’s -- >

<!ELEMENT title - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST title
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >

<!ELEMENT superscript - - (#PCDATA) >

<!ELEMENT subscript - - (#PCDATA) >

<!ELEMENT data - - (dataname? , datavalue*) >

<!ELEMENT dataname - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST dataname
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >

<!ELEMENT datavalue - - (datadescription* | reference* | (value , unit?)*)
>

<!ELEMENT datadescription - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST datadescription
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >

<!-- The directory, filename, suffix and NDATA are found in entities in a
static SGML/HyTime document. Dynamically built document must store this
information someplace else, here it is done with references to database
meta material. The filename attribute is filled during document
generation. Handling the link to external document also poses a problem
with dynamic document because the link is normally managed through
entities. There are two choises: either dynamically generate new entities
or hook into the process that deals actually reading the entity’s
contents. The latter is how Multidoc Pro does this: when the entity stored
in nmlist element is required, the parent reference element information is
looked instead of normal entity handling. -->
<!ELEMENT reference - - (ref.name? , nameloc) >
<!ATTLIST reference
id ID #REQUIRED
mediatype (SGML , NON-SGML) "SGML"
linkend IDREF #REQUIRED
HyTime NAME "clink"
directory IDREF #IMPLIED
filename CDATA #IMPLIED
suffix IDREF #IMPLIED
ndata IDREF #IMPLIED -- Defaults to SGML --
table CDATA #IMPLIED
column CDATA #IMPLIED >
```

```
<!ELEMENT ref.name - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST ref.name
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >

<!ELEMENT nameloc - - (nmlist) >
<!ATTLIST nameloc
id ID #REQUIRED
HyTime NAME "nameloc" >

<!ELEMENT nmlist - - (#PCDATA) >
<!ATTLIST nmlist
entity-info IDREF #REQUIRED
docorsub ENTITY #IMPLIED
nametype (element , entity) "element"
HyTime NAME "nmlist" >

<!ELEMENT value - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST value
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >

<!ELEMENT unit - - ((#PCDATA) | subscript | superscript)+ >
<!ATTLIST unit
content IDREF #IMPLIED
table CDATA #IMPLIED
column CDATA #IMPLIED >
```

Appendix C

Sample Database Mapping

This appendix shows the beginning of a sample database mapping file. The plan was that this format would have been replaced by a binary format. The Appendix B shows an SGML version that is both the display format and save format at the same time.

The DSN row in the beginning identifies the ODBC data source name which this mapping connects to. The KEY rows identify nodes, MAP and REL rows contain information about the actual mapping of the node. Nodes become SGML elements in the generated document instance.

```
DSN=LSAR SAMPLE
KEY=DATABASE:1
MAP=LSAR SAMPLE;0;0;LSAR SAMPLE;0;0()()
REL=
KEY=DATABASE:1,LEVEL:1
MAP=Projects;1;0;Projects;0;0()()
REL=Projects.Project ID = Systems.Project ID
KEY=DATABASE:1,LEVEL:1,TITLE:1
MAP=Project Name;2;0;Projects;0;0()()
REL=
KEY=DATABASE:1,LEVEL:1,LEVEL:1
MAP=Systems;2;0;Systems;0;0()()
REL=Systems.Documents ID = Documents.Documents ID;Systems.Documents ID =
Per Maint Documents.Documents ID;Systems.System ID = Units.System ID

...
```