A JAVA BASED COST MODELLING TOOL: STRUCTURE AND INTERFACING

Presented to the Graduate Council of
Southwest Texas State University
in Partial Fulfillment of
the Requirements

For the Degree

Master of SCIENCE

By

Mohammed Rahman, B.E.E.E.

San Marcos, Texas

August, 2000

Acknowledgements

All praise is due to Allah, the most Gracious and most Merciful.

I would like to thank Dr. Michael G. Wahl for his valuable guidance and advice during the entire thesis work. I also would like to thank my dear wife, Qumrun Nesa for her extreme patience, encouragement and support during the research work.

Mohammed Rahman

Southwest Texas State University

August, 2000

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

ACM	Abstract Cost Model
API	Application Process Interface
AWT	Abstract Windowing Toolkit
CAEc	Computer Aided Economic
CBE	Cost Bearing Entity
CMT	Cost Modelling Tool
CORBA	Common Object Request Broker Architecture
COTS	Components Off The Shelf
CPU	Central Processing Unit
CSS	Cascading Style Sheets
DFT	Design For Testability
DOM	Document Object Model
DTD	Data Type Definition
EVEREST	European Vanguard Effort on Research of Systems for Testing
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
IC	Integrated Circuit
JFC	Java Foundation Classes
MVC	Model View Controller
PCDATA	Parsed Character Data
RAD	Rapid Application Development
SAX	Simple API for XML
SGML	Standard Generalized Markup Language
URI	Universal Resource Identifier
XML	eXtensive Markup Language
XSL	eXtensive Style Language

I INTRODUCTION

Products with long lifetime tend to cause high costs of maintenance and repair. Often these post purchase expenses outweigh the cost of the product itself. Extensive research effort went into the optimization of design and manufacturing costs. However, little has been done to trim down field maintenance and support costs. In order to fill in this gap, faculty members and students from the University of Texas at Austin, Southwest Texas State University, and the University of Siegen, Germany started a project called LCCA (Life Cycle Cost Analysis). The goal of this project is to design a tool that will allow analyzing lifetime costs of a product during its design phase. This approach will lead to substantial reduction of costs and thereby increase the profit for the manufacturer as well as reduce the costs for the users.

Many tools have been developed, but they were either closely linked to a no longer existing design system or they were spreadsheets[11]. Both approaches lead to an inflexible solution that required reprogramming whenever the cost model changed. The goal of my research was to study the feasibility of developing a tool that is flexible, modular, works in a distributed environment, customisable, and easy to use.

This project progressed in several phases. The first phase of this project required intensive research encompassing the LCCA project and its goals, DFT (Design For Testability) and other mathematical models developed after several years of effort by numerous researchers. I studied a number of dissertations, magazine articles, and attended several group meetings with Dr. Anthony P. Ambler of the University of Texas and Dr. Michael Wahl of SWT, two of the leading researchers on LCCA project. The

second phase of the work was to study the previously developed tools, their merits, and come up with a set of requirements for the proposed Cost Modelling Tool. The third phase was to study the feasibility of developing a tool based on the concept proposed by Kaminski [14]. During this phase I studied available web technologies so that we can develop a tool that will support secured data processing over a distributed network. This included a thorough analysis of the capabilities of HTML, SGML, XML, and DTD. I also learnt about various XML tools such as "XML Generator", "Xerces-J", "Breeze XML Studio", and "Project X". There was a substantial learning curve in this phase of my research because I downloaded, and installed each of these tools, learnt how to use them by going through the tutorials, wrote some test code using example XML files and finally analysed against other tools.

Once we were sure of the possibility of developing a tool based on XML, DTD, SAX parser, and W3C DOM, we drew up the Cost Modelling Tool architecture in the fourth phase of our research. Simultaneously, we developed the Abstract Cost Model and all its corresponding sub models to create a template for the XML data structures. The fifth phase was to decide on a suitable Integrated Development Environment (IDE). During this phase I learnt to use JBuilder 3.5, Visual J++, Simplicity for Java, and Forte for Java. I made a comparative analysis of these tools and suggested the best IDE for this project. The following phases were: writing Java code for the CMT, testing the code against various XML data files, analyse memory usage, and deploying the tool in various environments.

This thesis consists of seven chapters. Chapter 1 gives a background of the LCCA project; previous cost modelling tools and their problems. Chapter 2 discusses basics of

economics modelling and system hierarchy. Chapter 3 discusses the requirement specification of the tool, the environment, and abstract models. Chapter 4 explains the merits of Java as a programming language and Java IDEs (Integrated Development Environment). Chapter 5 explores current state of XML, DTD, and XML parsers. Chapter 6 explains the Cost Modelling Tool architecture and the user interface. Finally, Chapter 7 concludes this thesis by opening doors to further research and enhancement of this tool.

1 Thesis Background

Design, Test and Economics are topics that are usually not discussed together. There is the well known wall between design and test which is difficult to cross, although throughout the years the wall has become significantly smaller. The gap between the technicians and the white-collar squad is much deeper due to quite different view of the product. The technicians of all colors at least have in common the view of an electronic part with electrical and timing specifications. The accountants have a much more abstract view of the part: 'This part costs $5 each and that is too much.' Obviously, these two groups speak a different language.

Looking at the result, the cost issues are essential because they sway the gain or loss of a company. Nevertheless, it's not so simple to optimize costs. There are many reasons for that. One reason is the different language spoken in the accounting and engineering departments. Here only a continuous training can help to overcome the communication problem.

A quite different problem is the size of a project. Even running a single person project can cause some confusion. The decisions made are often determined by preconceived ideas, even if all information is at hand. It gets of course worse when the project grows and many people are involved. As soon as the size of the team gets bigger than 7, one of the magic numbers, a hierarchy is required for reducing the communication time and keep people doing their work [2]. On the other hand, many design decisions are made which can produce only a local optimum, limited by the knowledge and the scope of the decision-maker.

Most of the products developed have a comparatively short lifetime. Very simple ones like electronic wristwatches are produced and sold and nobody even thinks about repairing these devices. High quality goods, e.g. high end stereo equipment, with a long lifetime are still repaired, although through the quality level we have achieved, repair is getting less important. In both cases costs occur mostly during the design and manufacturing stage, for the moment ignoring marketing and sales.

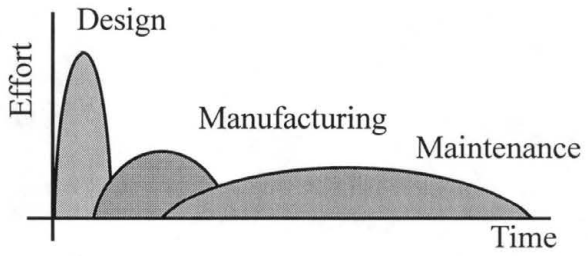

Figure 1 Financial effort over the life time [18]

Besides the product categories described above there are markets for long living goods. The first one is public transportation. Aeroplanes are maintained to achieve a lifetime of more than 25 years. Similar requirements are valid for rail bound systems. The

second category is formed by production facilities like power plants. The lifetime of one of these systems is quite high - at least 25 years. A third category with growing lifetimes is formed by military applications, where due to budget cuts the lifetime of existing goods are extended and new goods have to be planned for an even longer life span. In addition, all these systems require a high availability and reliability.

The costs for these long lasting goods have three parts: the design phase, where high costs occur over a short time period, the manufacturing phase, which is the time the product is sold on the market and finally the maintenance phase which is much longer. One of the major cost sources that are usually ignored is testing. On the IC (Integrated Circuit) level, it can be in the range of 30% - 50% of a component's manufacturing cost. Thus testing is one of the most crucial parts, strongly influencing the design, the manufacturing and the maintenance costs until the final end of use.

Testing has to be kept in mind during the specification phase to be able to deliver a product with sufficient quality. The DFT (Design for Testability) technique can be used to achieve this goal. Test equipment costs a lot of money (testers for processors are in the range of $3-5 Million), but we need it. Systems must be tested for preventive maintenance or because of a failure. The failure may be generated or falsely detected by using other, non-appropriate test equipment.

Testing is essential because the customer requirements towards quality and reliability of a product have to be met. For some time the idea of manufacturing a fault free product was discussed, but the reality showed that testing cannot be avoided.

Looking more into the technical side of a system the hierarchy mentioned for the human organization is of course part of the system, too. The hierarchy can be found in a

wide variety of systems. On the 'small scale' a single chip can consist of a CPU, memory, I/O devices, amplifiers, A/D and D/A converters and even micro mechanical parts. In the 'medium scale' you can find a system consisting of boards and chips. On the large scale a system might be an autopilot, consisting electronics and mechanical parts like ailerons, rudders or elevators.

The following figure shows this typical real life system in more detail as a collection of linked items. Ideally, the links are all the same type, but in reality many different views exist. Some of them are logic links, like the computer architecture hierarchy of CPU, memory, cache and I/O system. Other ones represent physical conducting links between CPU, SRAM, SDRAM - all running on the main board, which also physically caries them. Looking at the I/O again shows different types of modules: basic human communication elements, devices providing input on the location and the control elements, linked to the structure of the aeroplane.

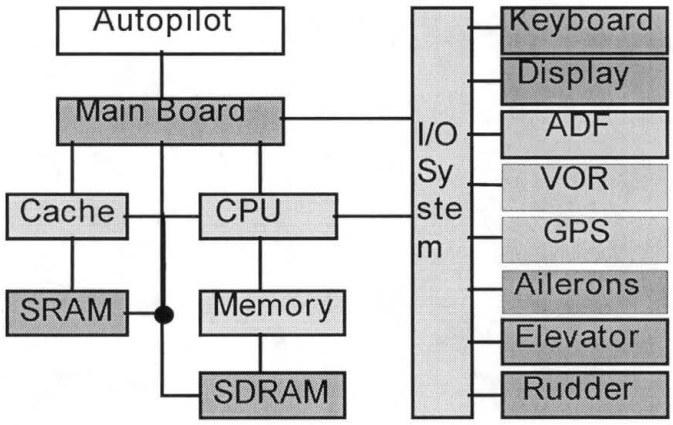

Figure 2 A more realistic system

In summary, the structural hierarchy of system, board and chip has to be linked with the three major phases of the lifetime of a product: Design, manufacturing and

maintenance. This leads to the following figure, which illustrates these areas and their relationships.

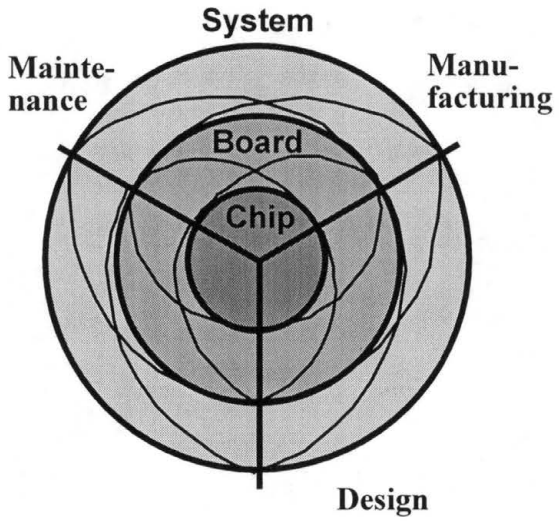

Figure 3 Hierarchies and time dependencies [18]

In the above diagram, all dimensions influence each other resulting in a set of arcs linking all levels and all life times. This graph, together with the illustration of a real system, clearly indicates that the problem of economics of design and test is not a trivial one. It reminds one of a multihued dragon, changing its colors continuously or, in a more abstract view, it is a multidimensional time dependent problem. For solving the problem there are no elf stones available and no silver bullets. From the technical perspective, there is no consistent cost model covering the wide range from design via manufacturing to maintenance. Local solutions exist, but still there is a lot of unknown space between these islands of knowledge and tools permitting cost modelling are simply not available. The following section examines the merit and the usefulness of the exiting tools.

2 Previously Developed Tools

Several tools have been developed during the history of cost modelling. There are two different approaches that have been followed: a spreadsheet oriented approach and a C-program based approach.

The previous cost modelling tools have been developed in the framework of a large project called EVEREST [7]. It was sponsored by the Commission of the European Community in the early '90s. The first kind of tool was based on spreadsheets. The idea behind this implementation was to have an easy to use model with a limited complexity. The current state of the spreadsheet software allows easy calculations and provides a huge variety of representations. These properties makes them well suited for business applications and better than ad hoc implementations.

The limitations of spreadsheets become obvious whenever the basic model is changing. A modification of the model requires an update of the spreadsheet. Unfortunately, the ability for documenting a spreadsheet so that it is easily maintained for a long time is limited. Besides that, the spreadsheets then too become slow when the number of variables becomes greater. It was observed that loading a spreadsheet database with several hundred parameters and some calculations could easily take minutes. Here the practicability becomes more important than the ease of programming. Besides, spreadsheets are designed to run on one machine only.

Another approach has been followed in the EVEREST project, too. The cost model for Asics that had been developed there was closely liked to the in-house CAD system. The interfaces of the CAD system were available and so the system accessed

many of the internal CAD data. The program was implemented in C language. The cost model was read in from a file. The model could be modified but due to its close link to the in-house CAD systems, the tool became useless for other applications.

Moreover, both the hardware and the operating system of the original software are no longer available. This is one of the key reasons why porting the code turned out to be really difficult. Another problem with this model was that the CAD system where the tool was directly linked to is no longer existent.

Part of the program was a simulation module, which allowed generating input value sets based on a Monte Carlo approach for determining the sensitivity of the individual parameters. For one analysis step 100,000 runs were performed, obviously a well suited approach for a tool running in batch mode, but not suited for an interactive tool.

II ECONOMICS MODELLING

Economics modelling is a technique, although seldom used, to deal with the key issues of a long lasting system. These long lasting systems, which are the focus of our work, have very high maintenance cost compared to their purchasing cost. For instance, the maintenance costs for a typical helicopter can be as high as 92% of the total system costs.

There are areas where costs are effectively optimized, particularly, in production lines for mass produced goods. This is comparatively easy to achieve because most of the information is available in house. It gets much more complicated as soon as the system consists of different elements, e.g. mechanical and microelectronic parts, which cause completely different requirements for test and maintenance. Besides, the information is far too complex to be handled by a single engineer.

Mechanisms are required for coordinating the work efficiently. System engineering standards are a basis for effective work, but most standards exist only in paper and are not really implemented. One of these standards is EIA [6].

1 System Hierarchy Levels

Digital system design constitutes a strict hierarchy of chips, boards and systems. Looking at the state of the art processors, the manufacturing costs split approximately half for manufacturing the silicon and half for the test. A set of various DFT methods is required, otherwise, the chips cannot be tested. Key information for the chip test level

comprise the test pattern for the basic modules, the structure according to the DFT approach (scan path chains, self test modules, etc.) and the DFT methods for the next higher level, e.g. the boundary scan (IEEE 1149.1). The test equipment usually is a standard digital test system or for analog components - an analog tester.

The board level requires DFT measures, too. From the old times access via test points is still in use, but due to the continuously growing complexity, boundary scan is a must today. Self-test is often implemented for the digital part, whereas such methods are missing for analog components or modules. For system testability, buses are provided, e.g. according to IEEE 1149.5.

The complete electronic system is usually tested on a special test system, reflecting the requirements of the digital and the analog parts. Using Components Off The Shelf (COTS), e.g. VXI components, drastically reduces the costs. Keeping the test systems running is a crucial point, because maintenance for these systems is not a marginal cost item.

Looking at a wider range immediately leads to a more general view of a system. Systems in general can have electronic, mechanical, hydraulic and pneumatic components, and all these partial systems can have a hierarchy as described for the electronic system. Nevertheless, there are significant differences: DFT methods are mostly used in digital systems, whereas in the other domains DFT is often not possible.

Besides, the requirements for complex systems change drastically. Chips and boards are usually analyzed for optimal design and production costs. However, the whole system must survive for a long time. Therefore, test systems must be at hand for the components over the complete lifetime of the product. The overall costs for a long lasting

system are dependent on the complete hierarchy - chip, board, electronic system, all other system components, as well as the cost of maintenance and repair.

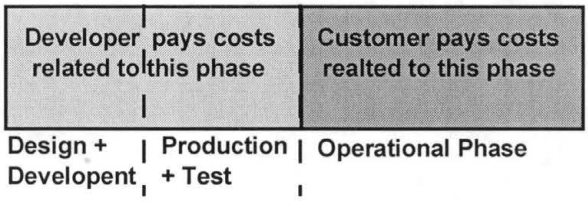

Figure 4 Developer and customer costs

The cost distribution between the developer and the user of a system is shown above. We studied the cost dependencies of the maintenance phase itself in more detail. The structure of the costs related to maintenance is shown in figure 5.

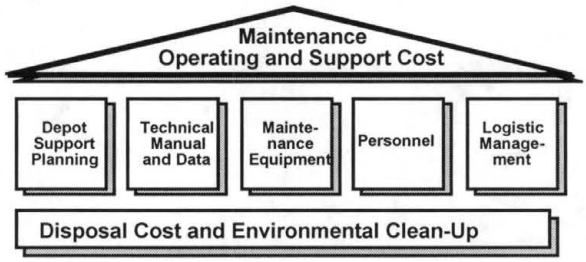

Figure 5 Columns of maintenance costs [4]

Every one of the five columns in the figure has strong impacts on the operational costs. In the depot support planning, we face 2 to 3 levels of maintenance dilemma where a decision of using an intermediate depot or not has to be made. The documentation of the product has an impact. Detail documentation can decrease the repair time. As for the maintenance equipment, we have two choices: "high tech equipment and intellectually disadvantaged people" or "intelligent engineers and less equipment".

The column of personnel decisions is not only a question of how well educated personnel but also how many personnel are needed to perform the maintenance. Logistic

management is a question of how to decrease the downtime of the product. The equipment maintenance and the personnel columns can not be truly separated because they have strong impact on each other. This makes the problem even more complex. DFT strategies can have impact on all columns of this figure.

The detailed analysis of the individual columns has shown that the complexity is extremely high because there are plenty of cyclic references in the cost model. Besides the pure complexity, there is another problem: the complete analysis would require information for all layers, starting with the chip level and ending at the system level. The information is present in most cases, but there are technical difficulties in accessing the information (a multitude of CAD tool interfaces are required) as well as organizational problems (who is willing to share his private data?). These two aspects together lead to the result that the development of an overall cost model is not feasible, at least not with our limited resources.

To cope with this complexity we have chosen to define a flexible cost model that can be expanded as necessary during the design, manufacturing and maintenance phase. Chapter III shows the modelling approach and gives some examples on how to describe a system with this model.

III TOOL REQUIREMENTS

The analysis of the previous tools showed that both approaches were not feasible for meeting today's requirements. The requirements are quite different, in particular as the intended use has changed and the view has become broader. The previous tools were intended for use within a single design group, whereas the new approach targets the systems' design flow by collecting information from a wide rage of different sources.

The requirements of the tool have grown significantly. The model cannot be assumed to be complete. The model will change and grow over the time. The hardware and software environment will no longer be constant and even differ in a single design flow. The data will be distributed and since there are different sources, the information must be protected. These items are discussed in more detail in the rest of this chapter.

1 Flexibility Of The Model

The discussion on economics modelling in the previous chapter clearly showed that it is difficult to generate a complete model. Thus we have to live with a model that will never be complete, but which will obviously not be constant, either. Thus we need a graphical model editor.

The core part of the model editor is the handling of the system hierarchy. Complex systems imply a hierarchy of chip, board, digital subsystem, and system. Linking other domains to the hierarchy leads to different paths toward a system. An

example is given in the following figure, where the hierarchy of digital system and a mechanical system are lined out.

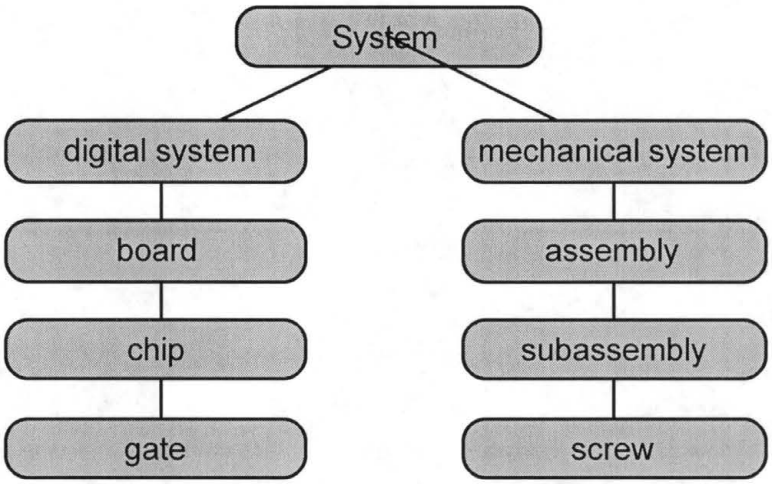

Figure 6 Two possible hierarchies: digital and mechanical

Obviously there are many more hierarchies possible, with a varying depth of hierarchy and mixed objects, even with objects having properties common to more than one hierarchy.

This leads to the development of a model, which is based on a generic entity, called *cell*. The name is borrowed from the IC design domain where a cell always is the basic block for the current level of hierarchy, independent of the complexity. A *cell* can have one or more properties, reflecting the different characteristic. This cell concept gives a high flexibility given the number of hierarchy levels and a mixture of properties of one *cell*. This concept also allows the dynamic extension of the model.

2 Hardware And Software Environment

The tool will be used within the system design flow. Therefore, different departments will have access to it - some will fill in specific information while others will refer to the information contained in the system. In the ideal case, a company-wide unified IT system is used, which could be the platform for implementing the cost modelling tool. However, this situation is usually not the case. The tool itself must be able to cope with a distributed user community.

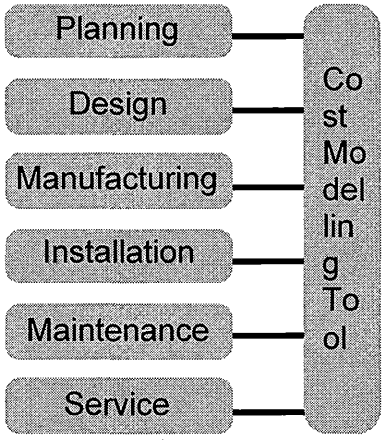

Figure 7 Departments with access to the cost model

Concerning the hardware there is only a very small probability that all departments use the same hardware. Usually Wintel PCs are used in the office domain, whereas in the design environment, quite often Unix workstations are in use. The operating system also varies in the same way: Windows and all its different implementations, Unix and its derivatives, MacOS and others. This variety led to the decision to use Java as the implementation language. Currently this is the language of the choice if multiple platforms must be dealt with and the development of separate versions is not possible.

The use of Java also allows the creation of a cost modelling applet that is downloaded on demand by the user. This concept has many advantages. The most important one is that a browser handles the user interface. Browsers are available on all platforms and when the tool is used via a web page, the page and the applet is downloaded. No installation of the tool is required at all and everybody will use the same version. Software updates will take place only on the central tool server.

3 Distributed Database

In the same way as the user community, the data itself is distributed. The data will not be available at one site and having a distributed database system as the foundation is not feasible for this project. Fortunately, the web makes it easy to deal with distributed data and the application is not time-critical.

The simple version is that the data is stored on the local server, together with the tool itself. The data can be accessed from the remote servers via the Intranet. The model is stored in a standardized file format, which supports links to external files. These external files can reside on the server, but they can be located on the local workstations, too. The link is established by using the URL of the file.

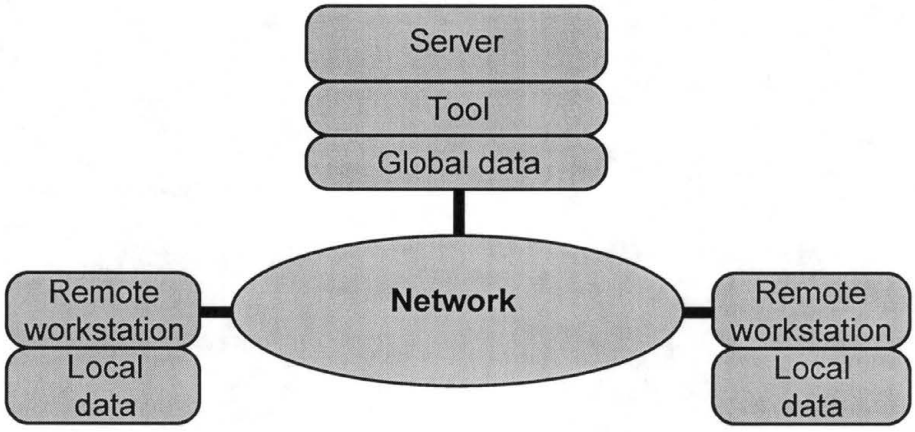

Figure 8 Data distributed over the global server and workstations

Restrictions to the data file nesting are that there must be exactly one file that represents the entry point to the model and of course there are no circular references allowed.

Up to now we have assumed that all information is owned by one organization and that the network is available in an Intranet. We have assumed, too, that the data connections are secure and that everybody who has access to the data is allowed to deal with it. However, for a real world application, we have to make sure that the access to the data is restricted. For increasing the security two mechanism have been introduced: explicit control of ownership and encryption.

The protection is not bound to the complete data set but linked to individual *cells*. The *cell,* as the core point of protection, allows fine-tuning of access control by keeping parts of the model protected while other *cells* are made available.

At the current state of the development, encryption has been introduced into the data model, but the encryption algorithm has not been decided on. Encrypted *cells* can be

accessed if owner identification matches the decryption key. The ownership control and the access right have been implemented.

We have defined three different categories of ownership: owner, user, and world. The *owner* of the model is a person or organization that has generated the data. A *user* is someone who must identify himself or herself by a name and password/decryption key. *World* is anybody reading the file but do not have to identify themselves. Each category has certain access rights: locked, evaluate, read, and modify.

Locked means that the *cell* is not accessible at all. As soon as the program reaches such a *cell* the construction of the tree is stopped and the cell referring to this cell becomes a leaf cell.

Evaluate means that the cell goes into the *cell* tree and that the program can evaluate the cost functions of this *cell*. The *cell* itself and all its contents remain invisible to the user.

Read means that the complete contents of the *cell* is visible and can be displayed by the cost modelling tool. A *cell* that is readable can always be evaluated. This mode opens all information to the user, but still the user cannot modify the model.

Modify means that the *cell* can be modified. This right includes reading and evaluating of the model. *Cells* can be changed and deleted from the *cell* tree and new *cells* can be linked to the current *cell*.

The owner of a data set always has the right to modify the cells and the model. The default values for user and world are locked. All access requires an explicit action from the owner. The following figure gives the right matrix.

	Locked	Evaluate	Read	Modify
Owner				default
User	default			
World	default			

Figure 9 Rights matrix

4 Abstract Cost Model

The abstract cost model is an initial attempt to develop a general cost model. It covers system costs over the various levels of hierarchy (chip, board, and system) as well as the product life phases (design, manufacturing, maintenance, and disposal). The target of the model is the cost optimization of long lasting systems.

The meta language used is XML, a mark-up language that allows describing the model. This language is also one of the key web languages and thus ideal, because today's tools are based on distributed data sources and applications distributed over the web.

The approach we are following is based on cost information modelling, followed by the evaluation of the model to determine the cost relevant parameters. The previously implemented approaches lack flexibility. As has been shown in the previous chapters, the modelling of a complex system requires a flexible solution. So we decided to develop a concept that is not based on a specific cost model but provides the means to describe a general cost model. The concept is depicted in the following figure.

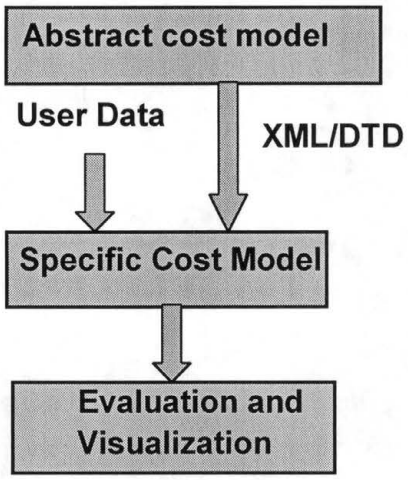

Figure 10 Basic software concept

Developing the abstract cost model (ACM) is a tedious work. It must allow the description of the general information about ownership and rights, the cost model itself, as well as information for the modelling tool so that the appearance of the data remains the same after save and restore. To describe the ACM, we used a style similar to Express-G [10] notation.

General information is required for 'housekeeping'. We need to know who is the generator of the data set, model version, which software and which software version, etc. The graph for this part is contained in the following figure. A key is provided since the data set can be protected.

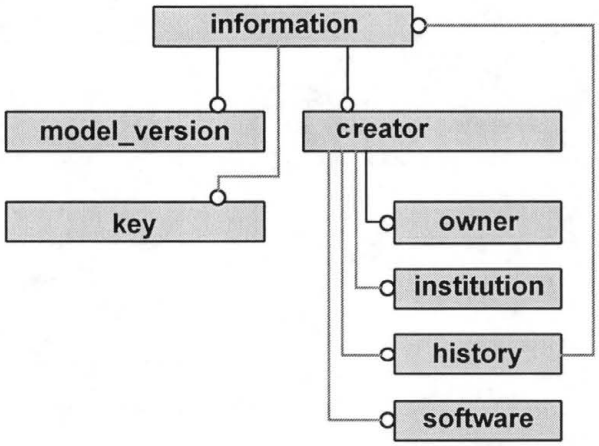

Figure 11 General information

5 Cost Bearing Entities

Cost Bearing Entities (CBE) are those entities that are associated with costs. To be able to collect information about different views of the model the entity can have one of five attributes: electronic, electric, mechanic, hydraulic and pneumatic. This set of attributes allows searching the cost tree for each category.

The costs themselves are separated in four different classes: design, manufacturing, maintenance, and disposal. The first two are the costs usually carried out by the designer and the last two are usually carried out by the users.

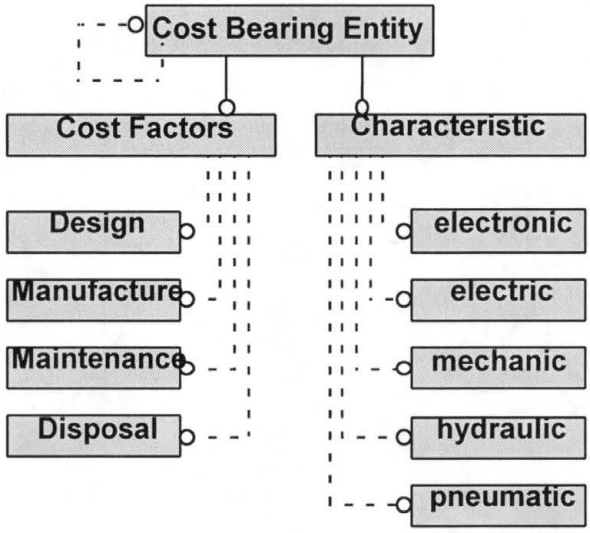

Figure 12 Cost Bearing Entity

A cost function is associated to each cost type. It may be that there are no costs at all associated at that level, but it may be a constant, too. Usually the costs will be described by an equation. The equation consists of global constants, e.g. the number of systems that are taken into account as well as constants like ***e*** and .

The functions themselves are described in postfix notation. The elements of the terms can be global constants, numbers, or other CBE's referenced by name. All standard arithmetic functions are supported. Following is the top-level graph of the model described above:

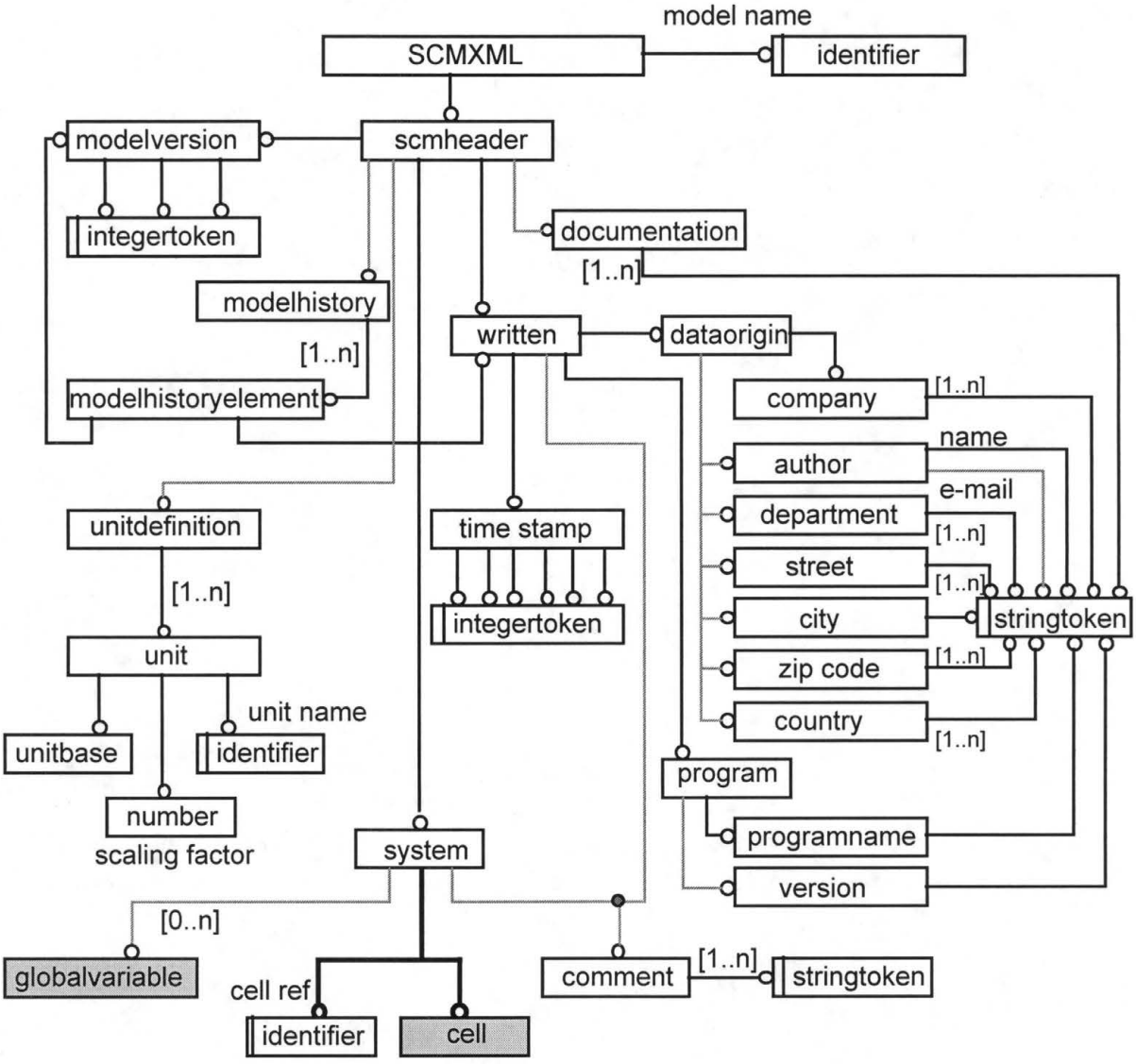

Figure 13 Top level model

Following is the cell model with extended cost functions:

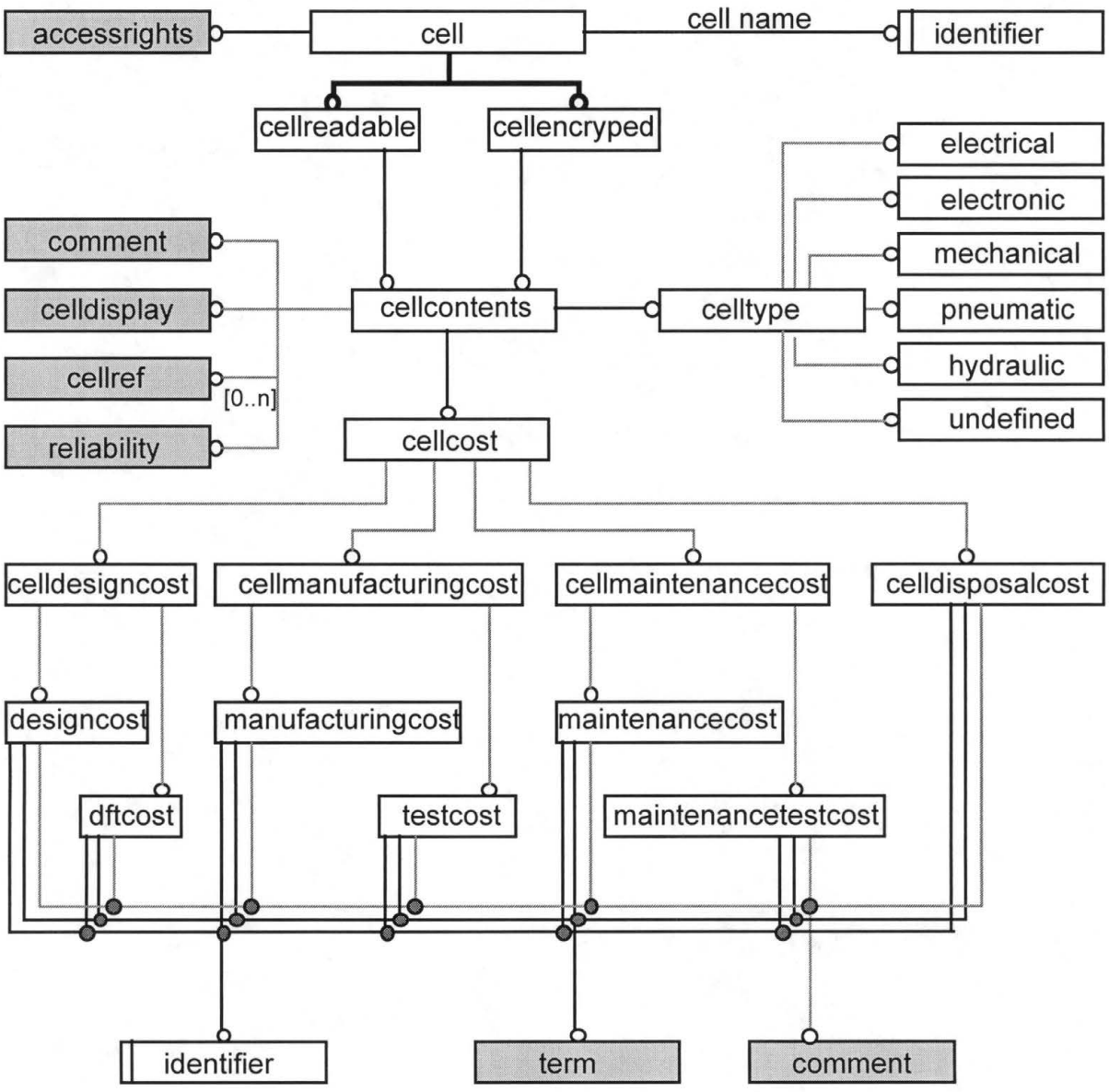

Figure 14 The cell model with extended functions

Appendix 3 provides definition of the terms used in figure 13 and 14.

6 The Modelling Tool

The cost modelling tool has several layers. The basic level is the file I/O level where the XML file is read and the data structure is created. The second layer is the

model management layer, where general data and CBEs can be graphically defined. The data including the information on how to visualize them can be stored back in the XML file.

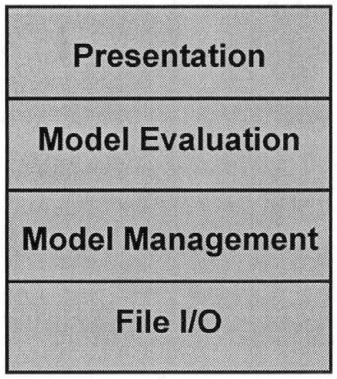

Figure 15 Layer structure of the cost modelling tool

The third layer is the model evaluation layer. Here a sensitivity analysis of the cost parameters can be performed. The visualization of the results will be performed in the fourth layer.

IV Java - Programming Language and Development Environment

The Java language developed and supported by Sun Microsystems is available on many platforms and because it is a virtual machine based language, the program itself as well as most parts of the libraries are the same on binary level on each system. This reduces portability problems and avoids the need to provide specific binaries for each platform a program should run on.

With the Swing-API, a platform independent, powerful, and modern graphical user interface library is available and with its pluggable look-and-feel it reduces the problems of platform independent programs for the user.

With its portability especially with Swing, Java is the language of choice for this project. The dynamic binding and eventually JavaBeans can be used for the Cost Modelling Tool to provide the dynamic addition of new objects for the modelling engine. Another important point for Java is that an XML-Parser is becoming part of Java's libraries, because of Sun's role in XML. Also Java has its benefits in development time, maintenance and reliability. These benefits outweigh the cost involved in requiring faster and better hardware, because it results in shorter implementation times. This is highly recommended related to the kind of this project with its very restricted development capacities.

Although it could not be assumed to be as stable as Java 1.1 with its several maintenance releases, the new Java 2 will be used. The necessary tool support is given by IDEs, such as, Borland's JBuilder 3.5, Sun's Forte for Java and Data Representation's

Simplicity for Java. Java 2 brings in several benefits like new basic classes, a just-in-time compiler on all platforms, CORBA, and enhanced Swing support.

1 Swing/JFC

The Swing API contained in the Java Foundation Classes is a platform independent, powerful, and modern graphical user interface library with a pluggable look-and-feel. Swing is written natively in Java and built on the JavaBeans technology. The original windowing toolkit, the abstract windowing toolkit (AWT), of the Java platform was just a thin wrapper around a platform specific GUI library. Thus, the programs always inherit the look and feel of that platform. This caused a lot of trouble for programmers to get their program run and look fine on different platforms.

For this project we used the latest Java 2 standard edition with JDK/JRE 1.2.2 and JFC/Swing 1.1.l.

2 The Integrated Development Environment

For developing a graphical user interface in Java an integrated development environment must be chosen. Several IDEs are available today: Borland's JBuilder, Sun's Forte for Java and Data Representation's Simplicity for Java are the most commonly used tools.

2.1 JBuilder 3.5

Borland's JBuilder 3.5 supports a variety of technologies including Java 2 platform, JFC/swing, JavaBeans, and CORBA. The JBuilder development environment

provides a single view for a variety of functions, such as file and project management, editing files, visual designing, compiling and debugging application. It consists of a window named *AppBrowser* that contains several panes and panels for performing these functions. JBuilder provides a variety of time saving wizards for application development. Code template and *CodeInsight* helps writing code. The UI designer is also quite powerful. Although JBuilder takes a while to load for the first time, compile and debug is quite fast. The Deployment wizard collects all the files needed to distribute a program. It can also archive them into a JAR file.

2.2 Simplicity for Java

Data Representation's Simplicity for Java is also a 100% Pure Java Rapid Application Development (RAD) tool for developing applications and applets using Java 2 Technology. With Simplicity for Java, developers can build Java software interactively through a visual model that is instantaneously updated to reflect any changes made to the program's source code without compiling. This lets developers create Java source code faster and with fewer errors. While coding, Simplicity executes Java source code on-the-fly, which allows developers to see their code in operation without lengthy code-compile-test cycles. Bugs and syntax errors instantly show themselves to the developer. The *Code Sourcerer* has a repertoire exceeding 50,000 different Java statements that can be chosen with a few mouse clicks.

2.3 Forte for Java

Sun Microsystems's Forte for Java IDE, formerly NetBeans Developer, is based on an open source code framework, is a new class of Java development tools. It is open,

modular, and easily extensible. It is written in the Java programming language and based on Swing components, and also cross-platform compatible. The core IDE can be extended to a complex and customized enterprise IDE. Sun has released Forte for Java just recently in June 5, 2000. Therefore, its capabilities and acceptability among the Java developers are yet to be seen.

There are many other IDEs exist but are not as popular or may be in the developing stage. JBuilder 3.5 is the IDE choice for our project because it has all the features we need, it is readily available at SWT, and it is widely used among the Java developers.

V Evolution of XML

The W3C consortium has developed a set of languages aimed at web applications. The best known is the HTML (Hyper Text Markup Language). This language allows describing a text format, but does not allow keeping any semantic meaning. That was the reason why SGML (Standard Generalized Markup Language) was developed. SGML is a meta language, designed for the specification of your own language. Since SGML is very general and it israther difficult to use. Subsequently, XML (eXtensive Markup Language) was defined as a subset of SGML and is the appropriate language for describing a data model.

1 Data Exchange

XML is a text-based markup language that is fast becoming the standard for data interchange on the Web. As with HTML, data is identified using tags (identifiers enclosed in angle brackets like this: <...>). Collectively, the tags are known as "markup". But unlike HTML, XML tags also informs what the data means, rather than how to display it. Where an HTML tag says something like "display this data in bold font" (...), an XML tag acts like a field name in a program. It puts a label on a piece of data that identifies it (for example: <message>...</message>).

In the same way that the field names are defined for a data structure, any XML tags may be used for a given application. Naturally, though, for multiple applications to use the same XML data, they have to agree on the tag names that they intend to use.

Here is an example of some XML data for a messaging application:

```
<message>
    <to>you@yourAddress.com</to>
    <from>me@myAddress.com</from>
    <subject>XML Is Really Cool</subject>
    <text>
        How many ways is XML cool? Let me count the ways...
    </text>
</message>
```

The tags in this example identify the message as a whole, the destination and sender addresses, the subject, and the text of the message. As in HTML, the `<to>` tag has a matching end tag: `</to>`. The data between the tag and its matching end tag defines an element of the XML data. Note, too, that the content of the `<to>` tag is entirely contained within the scope of the `<message>...</message>` tag. It is this ability for one tag to contain others that give XML its ability to represent hierarchical data structures.

Once again, as with HTML, white space is essentially irrelevant, so the data can be formatted for readability and yet still be processed easily with a program. Unlike HTML, however, data can be easily searched in XML, because the XML tags identify the content of the data, rather than specifying its representation.

2 Tags and Attributes

Tags can also contain attributes - additional information included as part of the tag itself, within the tag's angle brackets. The following example shows an email message structure that uses attributes for the "to", "from", and "subject" fields:

```
<message to="you@yourAddress.com" from="me@myAddress.com"
        subject="XML Is Really Cool">
        <text>
                How many ways is XML cool? Let me count the ways...
        </text>
</message>
```

As in HTML, the attribute name is followed by an equal sign and the attribute value, and multiple attributes are separated by spaces. Unlike HTML, however, in XML commas between attributes are not ignored - if present, they generate errors.

3 Well-formness

One really big difference between XML and HTML is that an XML document is always constrained to be well-formed. There are several rules that determine when a document is well-formed, but one of the most important is that every tag has a closing tag. So, in XML, the `</to>` tag is not optional. The `<to>` element is never terminated by any tag other than `</to>`. Another important aspect of a well-formed document is that all tags are completely nested. So you always must have nesting like this

`<message>...<to>...</to>...</message>`,

but never unbalanced tags like

<message>...<to>... </message>...</to>.

A complete list of requirements is contained in the list of XML Frequently Asked Questions (FAQ) on the w3c "Recommended Reading" list at http://www.w3.org/XML/.

4 The XML Prolog

An XML file always starts with a prolog. The minimal prolog contains a declaration that identifies the document as an XML document, like this:

<?xml version="1.0"?>

The declaration may also contain additional information, like this:

<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

The XML declaration is essentially the same as the HTML header, <html>, except that it uses <?..?> and it may contain the following attributes:

Version identifies the version of the XML markup language used in the data. This attribute is not optional.

Encoding identifies the character set used to encode the data. "ISO-8859-1" is "Latin-1" the Western European and English language character set. (The default is compressed Unicode: UTF-8.)

Standalone tells whether or not this document references an external entity or an external data type specification. If there are no external references, then "yes" is appropriate

5 XML Processing Instructions

An XML file can also contain processing instructions that give commands or information to an application that is processing the XML data. Processing instructions have the following format:

<?target instructions?>

where the *target* is the name of the application that is expected to do the processing, and *instructions* is a string of characters that embodies the information or commands for the application to process.

Since the instructions are application specific, an XML file could have multiple processing instructions that tell different applications to do similar things, though in different ways. The XML file for a slideshow, for example, could have processing instructions that let the speaker specify a technical or executive-level version of the presentation. If multiple presentation programs were used, the program might need multiple versions of the processing instructions.

6 Advantage of XML over HTML

There are a number of reasons for XML's surging acceptance. The following sections lists a few of the most prominent advantages of XML over HTML.

6.1 Plain Text

Since XML is not a binary format, it makes it easy to create and edit files with anything from a standard text editor to a visual development environment. That makes it

easy to debug programs, and makes it useful for storing small amounts of data. At the other end of the spectrum, an XML front end to a database makes it possible to efficiently store large amounts of XML data as well. So XML provides scalability for anything from small configuration files to a company-wide data repository.

6.2 Data Identification

XML identifies what kind of data there is, not how to display it. Because the markup tags identify the information and break up the data into parts, an email program can process it, a search program can look for messages sent to particular people, and an address book can extract the address information from the rest of the message. In short, because the different parts of the information have been identified, they can be used in different ways by different applications.

6.3 Stylability

When display is important, the stylesheet standard, XSL, gives the ability to dictate how to portray the data. For example, the stylesheet for:

<to>you@yourAddress.com</to>

can say:

1. Start a new line.
2. Display "To:" in bold, followed by a space
3. Display the destination data.

Which produces:

To: you@yourAddress

Of course, the same thing can be done in HTML, but it will not allow processing the data with search programs and address-extraction programs and the like. More importantly, since XML is inherently style-free, one can use a completely different stylesheet to produce output in postscript, TEX, PDF, or some new format that hasn't even been invented yet. That flexibility amounts to what one author described as "future-proofing" your information. The XML documents authored today can be used in future document-delivery systems that haven't even been imagined yet.

6.4 Inline Reusabiliy

One of the nicer aspects of XML documents is that they can be composed from separate entities. This can be done with HTML, but only by linking to other documents. Unlike HTML, XML entities can be included "in line" in a document. The included sections look like a normal part of the document - the whole document can be searched at one time or downloaded in one piece. That provides the ability to modularize documents without resorting to links. Additionally, a section can be single-sourced so that an edit to it is reflected everywhere the section is used, and yet a document composed from such pieces looks for all the world like a one-piece document.

6.5 Linkability

Thanks to HTML, the ability to define links between documents is now regarded as a necessity. This initiative lets XML files define two-way links, multiple-target links, "expanding" links (where clicking a link causes the targeted information to appear inline), and links between two existing documents that are defined in a third.

6.6 Easily Processed

Regular and consistent notation makes it easier to build a program to process XML data. For example, in HTML a <dt> tag can be delimited by </dt>, another <dt>, <dd>, or </dl>. That makes for some difficult programming. But in XML, the <dt> tag must always have a </dt> terminator, or else it will be defined as a <dt/> tag. That restriction is a critical part of the constraints that make an XML document well formed. Otherwise, the XML parser won't be able to read the data. And since XML is a vendor-neutral standard, one can choose among several XML parsers, any one of which takes the work out of processing XML data.

6.7 Hierarchical Structure

Finally, XML documents benefit from their hierarchical structure. Hierarchical document structures are, in general, faster to access because you can drill down to the part you need, like stepping through a table of contents. They are also easier to rearrange, because each piece is delimited. In a document, for example, you could move a heading to a new location and drag everything under it along with the heading, instead of having to page down to make a selection, cut, and then paste the selection into a new location [22].

7 Data Type Definition

Data Type Definitions (DTDs) are a means of establishing a schema for XML documents. DTDs originated in SGML and serve as the standard schema mechanism for validating SGML documents. A big benefit of using DTDs as schemas is that existing

SGML tools can be easily modified to support XML because they already support DTDs in SGML. Following is an example of a DTD for an XML addressbook:

```
<!ELEMENT addressbook (contact)+>
<!ELEMENT contact (name, address+, city, state, zip, phone, email, web, company)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT address (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT state (#PCDATA)>
<!ELEMENT zip (#PCDATA)>
<!ELEMENT phone (voice, fax?)>
<!ELEMENT voice (#PCDATA)>
<!ELEMENT fax (#PCDATA)>
<!ELEMENT email (#PCDATA)>
<!ELEMENT web (#PCDATA)>
<!ELEMENT company (#PCDATA)>
```

The above DTD reveals just the basic mechanics of how a document model is defined using DTD syntax. The content model for the contact element is provided as a list of elements enclosed in parentheses. The plus sign (+) following the addressbook element indicates that it can occur more than once. The phone element also includes a content model that lists the child elements of phone. In this case the question mark (?) indicates that the fax element is optional, but if appears it must appear only once. All of the remaining elements in the DTD are listed as containing #PCDATA. PCDATA stands for parsed character data.

It is important to note that data type constraining is not supported in DTDs. To use data type constraining, we must create a schema that adheres to the XML-Data specification, such as XML Schema.

8 XML Parsers

An XML parser is a program that reads or interprets XML documents. There are a wide variety of XML parsers available today. Most XML parsers can operate as either validating or non-validating parsers, depending on whether or not a document requires validation. Additionally, most of the available XML parsers are designed as software components that can be plugged into an application. The following is a list of some of the more popular XML parsers available as of this writing [13]:

- Sun's Project X Parser for Java
- The Lark and Larval XML Parsers for Java
- Datachannel's XML Parser for Java
- IBM's XML Parser for Java
- The Oracle XML Parser for Java
- Microsoft's Aelfred XML Parser for Java
- IBM's XML Parser for C++
- The XML Parser for C++ Builder

Cost Modelling Tool uses Sun's Project X Parser (JAXP). It has several advantages over other parsers. It allows parsing of XML 1.0 documents, supports optional validation for "well-formed" of "valid" XML code, supports SAX 1.0 API and XML's Namespaces. Sun's XML parser is the fastest parser available to date that has passed over 500 industry benchmark tests [22]. Additionally, this program offers support

for building and manipulating XML tree-structured data. It conforms to W3C DOM Level 1 recommendations with enhancements.

9 Breeze XML Tool

Breeze is an alternative to the DOM. It creates Java classes that encapsulate XML parsing and validation and which have methods that map to XML data elements and attributes. Basically, Breeze creates a custom API for an application-specific XML structure and gives a toolkit for transporting files. In other words, Breeze allows creating Java objects that encapsulate the parsing and loading of an XML document into Java classes. In order for us to use Breeze, these classes then have to be accessed to populate the JTree. Subsequently, the toXML() and fromXML() methods can be used on the Breeze object to get streams of XML into and out of these objects. First we liked this approach, but after a through evaluation of the tool, this approach turned out to be quite cumbersome.

Breeze may be good for application that uses specific data model. However, our data model is flexible, therefore, the DOM approach as described in the following chapter will work better for us.

VIII SOFTWARE HIERARCHY

The CMT software is based on MVC (model-view-controller) architecture. The data model, the tree view, and the menu operations are three separate entities. Changes in one will not affect the other. Data is extracted from the XML file and stored in the memory as a DOM. A JTreeModel is created from this DOM. The view is based on JTree, a swing component of Java. The following figure depicts the hierarchy.

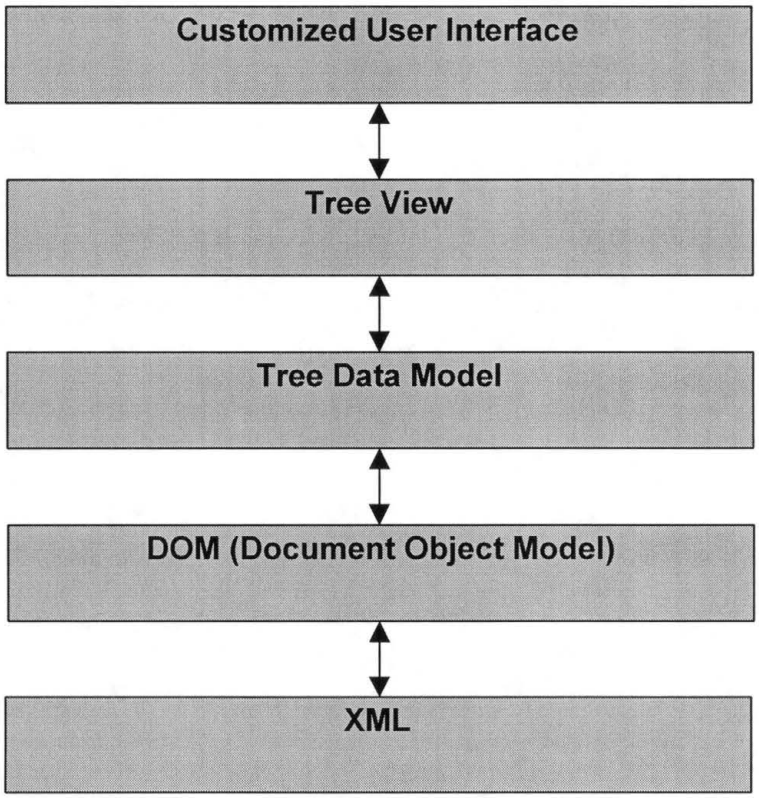

Figure 16 Cost Model architecture – top-level view

Users can change the tree using various menu operations. These changes are then propagated to the model and to the data. Now if the new tree is saved, a new XML file

will be generated that reflects the changes made by the user. The following sections will explain the operations in detailed.

1 MVC Architecture

The Cost Model Tool was designed using MVC (Model-View-Controller) architecture. The program creates and manages an association between three collaborating elements:

- View: This comprises the visible GUI that the user sees
- Model: This is the abstraction used in the application logic to represent the nature and state of the visual objects that are presented on the screen to the user.
- Controller: This component enables the model and view components to communicate. It also allows the model to be updated to reflect changes incurred by interaction with the user.

The job of the view object is to manage GUI issues such as rendering of the display components, layout, redrawing, event handling, and so on. These tasks are heavily focused on presenting a meaningful representation of the internal state of the application to the user.

The application model has a different job. It must create and maintain the internal application logic state associated with whatever entities the program represents or simulates. The GUI represents a visual analogy of the nature and internal state of the model.

The controller is really the part of the application that forms a communication path between the model and view. It allows the view and model to communicate changes in their state to one another.

An external entity other than the GUI can initiate a change to the state of the model. In that case, the model makes a request to update the view object. Another scenario is that the user performs some action that requires a change to the model's state. The GUI reflects this change, but it is really the model that tracks the change.

In either case, the controller is the medium through which the model and view communicate. The view translates the abstraction employed by the model into a form suitable for the user. The model, therefore, is the heart of the application logic and state.

2 Parsing an XML Document

The CMT program parses XML document using Sun's JAXP parsers. The following sections describe the technologies that JAXP is built on.

2.1 The JAXP APIs

The JAXP APIs, contained in the jaxp.jar file, are comprised of the javax.xml.parsers package. This package contains two factory classes: *SAXParserFactory* and *DocumentBuilderFactory* that create a SAX parser and a *DocumentBuilder*, respectively. The *DocumentBuilder*, in turn, creates DOM-compliant *Document* object. The remainder of this section shows how those APIs relate to each other in the Cost Model application.

2.2 An Overview of SAX and DOM

SAX and DOM APIs are defined by XML-DEV group and by the W3C, respectively. The libraries that define those APIs are included in the parser.jar file. The "Simple API" for XML (SAX) is the event-driven, serial-access mechanism that does element-by-element processing. The API for this level reads and writes XML to a data repository or the Web. The DOM API provides a relatively familiar tree structure of objects. The DOM API can be used to manipulate the hierarchy of application objects it encapsulates. The DOM API is ideal for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

On the other hand, constructing the DOM requires reading the entire XML structure and holding the object tree in memory, so it is much more CPU and memory intensive. For that reason, the SAX API will tend to be preferred for server-side applications and data filters that do not require an in-memory representation of the data.

2.3 The SAX APIs

The *SAXParserFactory* generates an instance of the parser, which parses the XML text. As the data is parsed, the parser invokes one of several *callback* methods defined by the interfaces *DocumentHandler*, *ErrorHandler*, *DTDHandler*, and *EntityResolver*.

Here is a summary of the key SAX APIs:

SAXParserFactory

A *SAXParserFactory* object creates an instance of the parser determined by the system property, javax.xml.parsers.SAXParserFactory.

Parser

The org.xml.sax.Parser interface defines methods like *setDocumentHandler* to set up event handlers and *parse*(URL) to actually do the parsing. This interface is implemented by the *Parser* and *ValidatingParser* classes in the com.sun.xml.parser package.

DocumentHandler

Methods like *startDocument, endDocument, startElement,* and *endElement* are invoked when an XML tag is recognized. This interface also defines methods *characters* and *processingInstruction*, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

ErrorHandler

Methods *error*, *fatalError*, and *warning* are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). Sometimes, the application may be able to recover from a validation error. Other times, it may need to generate an exception. To ensure the correct handling, the application needs to supply its own error handler to the parser.

DTDHandler

Methods defined in this interface are invoked when processing definitions in a DTD. This interface is extended by the com.sun.java.xml interface *DtdEventListener*, which adds methods like *startDtd* and *endDtd.*

EntityResolver

The *resolveEntity* method is invoked when the parser must identify data identified by a URI. In most cases, a URI is simply a URL, which specifies the location of a document, but in some cases the document may be identified by a URN -a *public identifier*, or name, that is unique in the web space. The public identifier may be specified in addition to the URL. The *EntityResolver* can then use the public identifier instead of the URL to find the document, for example to access a local copy of the document if one exists.

A typical application provides a *DocumentHandler*, at a minimum. Since the default implementations of the interfaces ignore all inputs except for fatal errors, a robust implementation may want to provide an *ErrorHandler* to report more errors or report them differently.

2.4 SAX Packages

Table 1 shows how the SAX parser is defined by different packages.

Table 1 Packages that define SAX parser

Package	Description
org.xml.sax	Defines the SAX interfaces. The name "org.xml" is the package prefix that was settled on by the group that defined the SAX API. This package also defines *HandlerBase* - a default implementation of a base class for the various "handlers" defined by the interfaces, as well as an *InputSource* class, which encapsulates information that tells where the XML data is coming

	from.
org.xml.sax.helpers	This package is part of SAX. It defines the *ParserFactory* class, which lets us acquire an instance of a parser either by specifying a name string or by using the value defined by the org.xml.sax.parser system property.
javax.xml.parsers	Defines the *SAXParserFactory* class, which returns the SAXParser. Also defines the *ParserConfigurationException* class for reporting errors.
com.sun.xml.parser	Contains the Java XML parser (com.sun.xml.parser.Parser), validating parser (com.sun.xml.parser.ValidatingParser), and entity resolver. The fully qualified name of either parser can be sent to the parser factory to obtain an instance of that parser. The *nonvalidating* parser generates errors if a document is not well formed, and does some processing of the DTD (if present) but does not check to make sure that the document obeys all of the constraints defined by the DTD. The validating parser, on the other hand, checks to make sure that the document obeys all such constraints.

2.5 The Document Object Model (DOM) APIs

The javax.xml.parsers.DocumentBuilderFactory class can be used to get a *DocumentBuilder* instance, and thereby produce a *Document* (a DOM) that conforms to the DOM specification. The builder, in fact, is determined by the System property, javax.xml.parsers.DocumentBuilderFactory, which selects the factory implementation that is used to produce the builder. The platform's default value can be overridden from the command line.

The builder's *newDocument()* method can create an empty Document that implements the org.w3c.dom.Document interface. Alternatively, the builder's *parse* methods can be used to create a *Document* from the existing XML data.

2.6 DOM Packages

The Document Object Model implementation is defined in the following packages listed in table 2:

Table 2 Packages that define the Document Object Model

Package	Description
org.wc3.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the *DocumentBuilderFactory* class and the *DocumentBuilder* class, which returns an object that implements the W3C Document interface. The

	factory that is used to create the builder is determined by the javax.xml.parsers system property, which can be set from the command line or overridden when invoking the *newInstance* method. This package also defines the *ParserConfigurationException* class for reporting errors.
com.sun.xml.tree	Sun's Java XML implementation of the DOM libraries, including the *XmlDocument*, *XmlDocumentBuilder*, and *TreeWalker* classes.

2.7 Cost Modelling Tool Implementation

This section shows how the Cost Modelling Tool implementation combines the SAX and DOM APIs. In this implementation, the DOM API builds on the SAX API as shown in figure 17. The Cost Modelling Tool uses the SAX libraries to read-in XML data and constructs the tree of data objects that constitutes the DOM.

The section of the diagram inside the wavy lines shows what Cost Modelling Tool implementation does when it parses existing XML data. The default *DocumentBuilder* creates an object, which implements the SAX *DocumentHandler* interface. It then hands that object to the SAX parsers. When the input source is parsed, the *DocumentHandler* creates a *Document* object. This object is then converted to a *TreeModel* by using an adapter called: *DomToTreeModelAdapter*

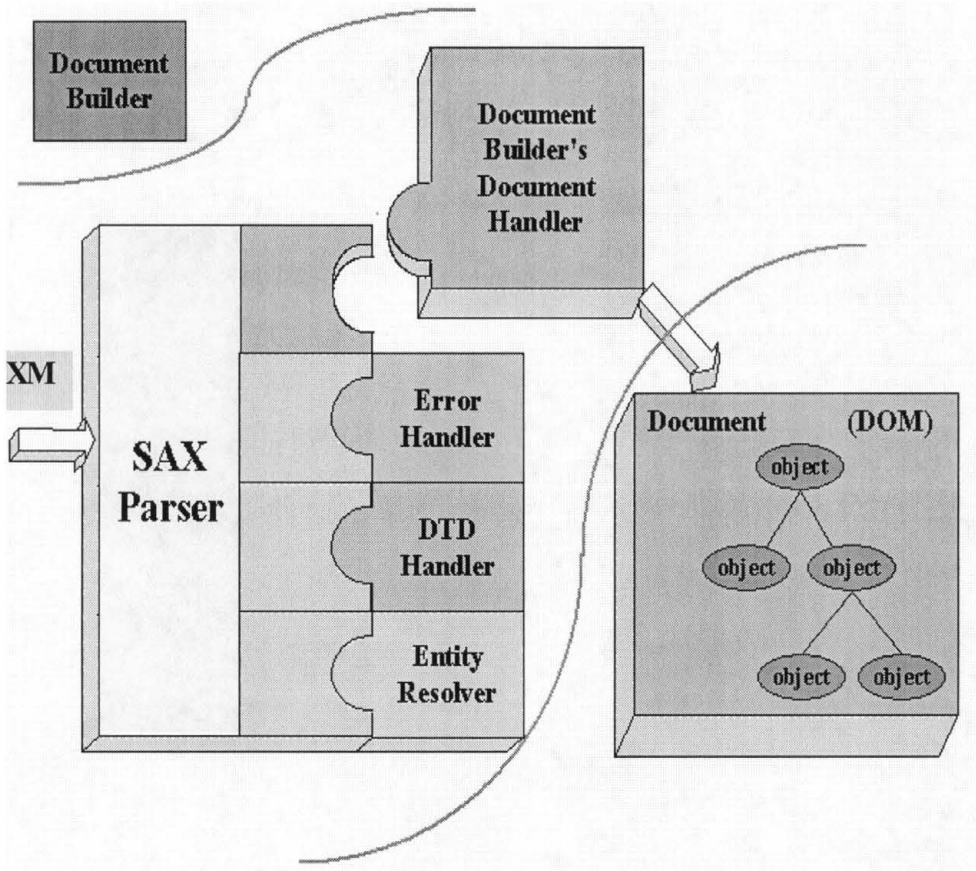

Figure 17 Parsing an XML document

3 Event Handling

After creating a model, a JTree object is instantiated. This object uses the model by invoking the *JTree (TreeModel)* constructor. Then the program registers event listeners with the *JTree* component to handle the events it is interested in. When the program handles a user event, it makes two calls: one to the underlying data structure to make the change, and another to the *TreeModel*-adapter to report the change. The *TreeModel* object then notifies the JTree of the change.

The diagram below shows the steps needed to create a JTree object. They work like this:

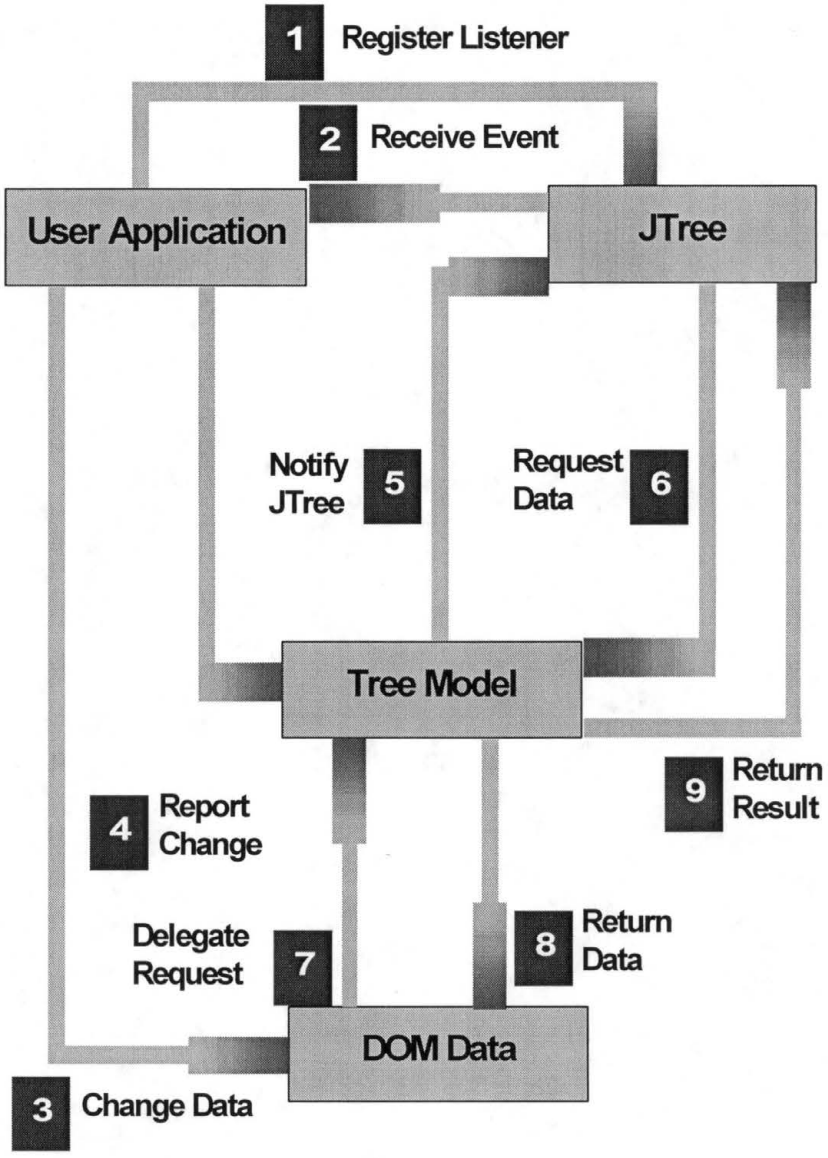

Figure 18 Cost Modelling Tool architecture – detail view

4 User Interface

Cost Model user interface is simple. It demonstrates the capabilities of the model. Based on the needs of the users, this interface should be customized. Figure 19 shows the user interface after it is invoked for the first time. The left pane displays the XML data in a view similar to Windows file explorer. The tree may be expanded or collapsed using the mouse. If any of the nodes is selected on the left pane, the corresponding data or text is displayed on the right editor panel.

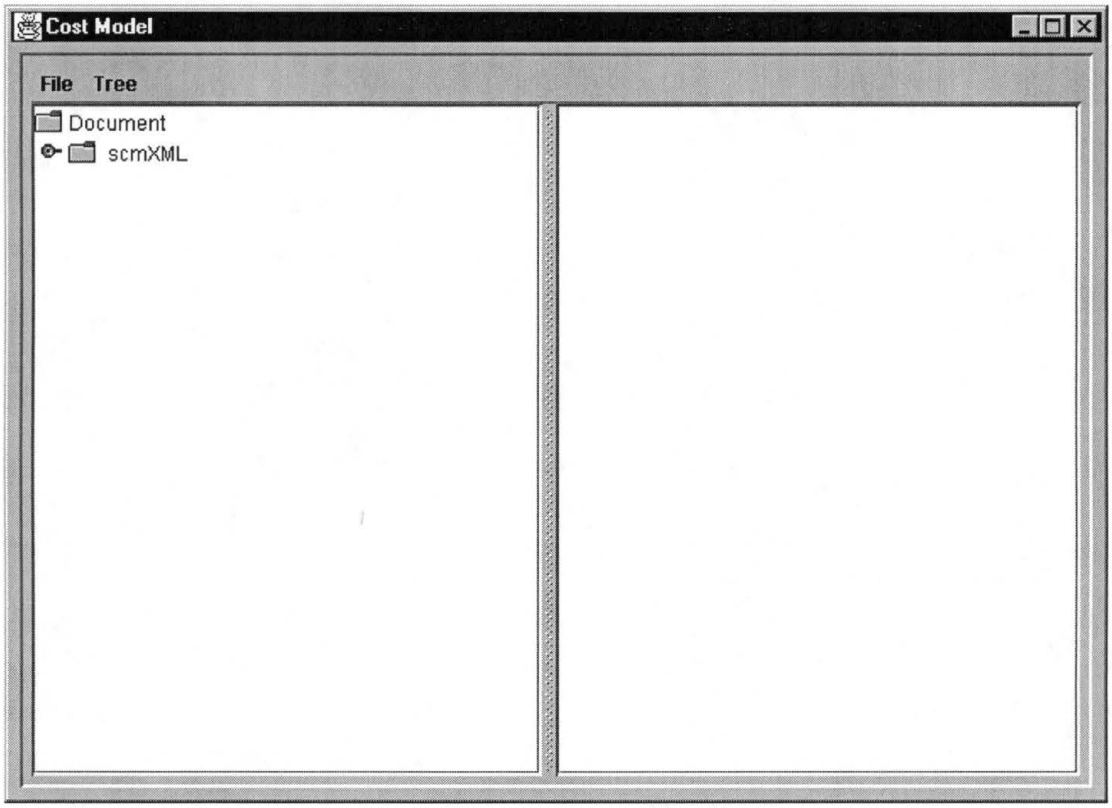

Figure 19 Cost Model Tool User Interface

The "File" menu has two options: "Save" and "Exit". "Save" operation generates a new *XmlDocument* from the current DOM document. This new document is then used to overwrite the old XML file. Note that writing a DOM is not standard until W3C DOM

level 3 which is at this moment being reviewed as a draft. Therefore, we have used Sun's ProjectX functions for demonstration purposes. Once DOM level 3 standard is approved, these functions should be replaced by their new counterparts. "Exit" menu closes the program thus frees up the memory used by the DOM and *JTreeModel*.

The "Tree" menu offers four operations: "Add", "Insert", "Reload", and "Remove". "Add" and "Insert" operations have not been implemented. "Reload" action serves as refreshing the tree's view to the underlying data model. Finally, "Remove" action removes the currently selected node from the tree, updates the DOM and informs the model of change.

The following figure shows the expanded tree created from Example.xml data:

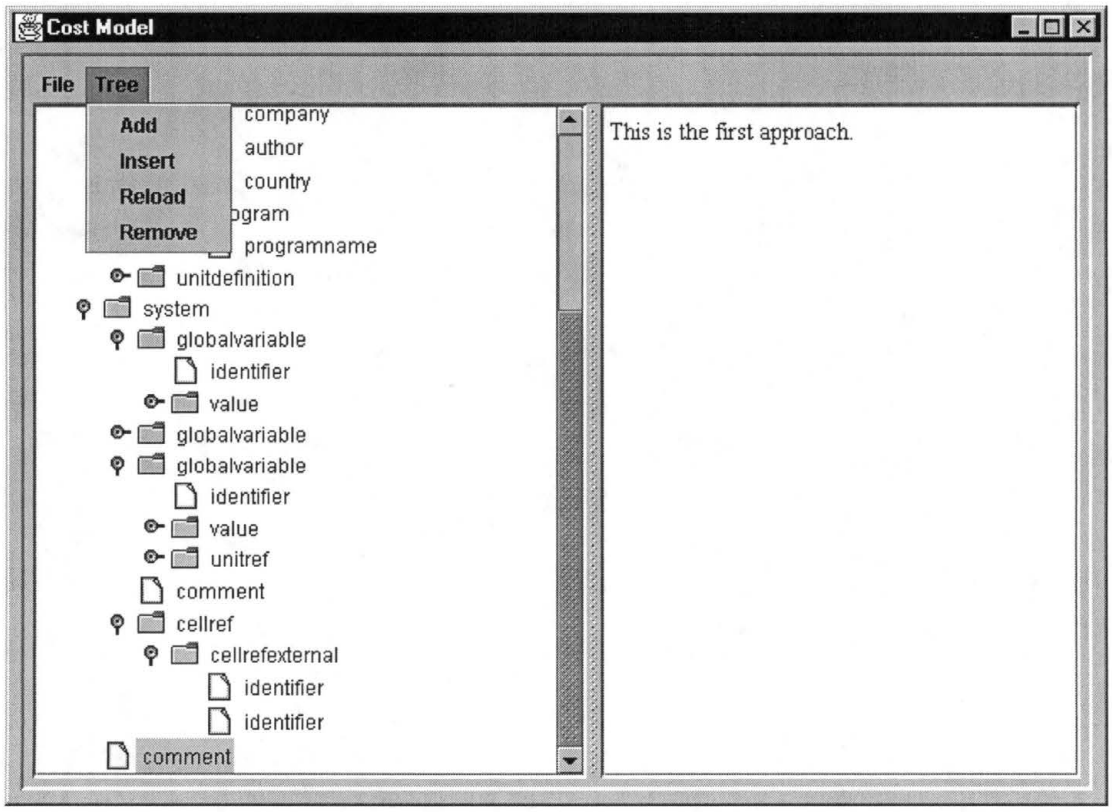

Figure 20 Cost Model Tool Operations

IX CONCLUSION

The CMT (Cost Modelling Tool) developed during the course of this project is a breakthrough for the LCCA (Life Cycle Cost analysis) project. It bridges the gap between the cost modelling concept and its actual implementation in the real world. It allows maintaining a gradually growing model so that the model itself can be adapted to the evolving technology and the increasing level of detail. Additionally, this tool will be able to analyze costs and thereby help finding the right directions to meet the aim of optimal costs.

XML was used as the carrier of data for the eventual web application for several reasons. XML is relatively simple, and very flexible. It has many uses yet to be discovered - we are just beginning to scratch the surface of its potential. It is the foundation for great many standards yet to come, providing a common language that different computer systems can use to exchange data with one another. As each industry-group devise standards for what they want to say, computers will begin to link to each other in ways previously unimaginable [17].

Although this thesis project has made a significant contribution to the development of the Cost Modelling Tool, the work is not complete. For the time being we have decided to define the data types according to the DTD but not dynamically. Not all editing functions are implemented because the intent of this thesis was to investigate the viability of core implementation of a concept and to provide a basis for further development into a final product. Appendix A contains a list of items that need to be implemented in the next phase of the project.

This project builds a very solid foundation by bringing together the new ideas of Java as a programming language, XML as the format for describing the data, DTD as the way for defining the data entities and the concept of a distributed web based Computer Aided Economic (CAEc) tool. The state of the art tool environment is a breakthrough in the commercial application of LCCA concept. The thesis also demonstrated that it is possible to perform software development in parallel to the data definition and even across the Atlantic.

Appendix A

Further Implementation

The following features need to be implemented in the Cost Modelling Tool in order to get a fully functional tool:

Editing the tree

The "Add" operation under "Tree" menu is not implemented. It can be implemented in different ways. The user selects a node and then chooses "Add" menu item. The program should then open a dialog box with list boxes, radio buttons, and text fields relevant to the node selected. Upon entering the input, the program will add the node to the tree and update the DOM. Other editing operations can also be added such as: cut a node, paste a node, move a node, rename a node, edit the content of a node, etc.

Converting the application to an applet

In order for the CMT to be deployed in the web environment, the application must be converted into a Java applet. However, in Java-enabled browsers, untrusted applets cannot read or write files at all. By default, downloaded applets are considered untrusted [1]. There are two ways for an applet to be considered trusted. Firstly, if the applet is installed on the local hard disk, in a directory on the CLASSPATH used by the program that runs the applet. Usually, this is a Java-enabled browser, but it could be the

appletviewer, or other Java programs that know how to load applets. Secondly, if the applet is signed by an identity marked as trusted in the identity database.

Customizing the User Interface

The UI needs to be designed based on the uses. The CMT should be used as an SDK and the UI will be designed according to the need of the users. The MVC architecture of CMT already allows such implementation. For example, the CMT UI for the manager, architect, and developer may be different due to the nature of their use of CMT.

Using XML Schema instead of DTDs

One of the big complaints about DTDs is that they rely on a specialized syntax for describing the structure of XML vocabularies. Furthermore, data type constraining in not supported in DTDs. On the other hand, XML schemas are more powerful and provide advanced features such as open content models, namespace integration, and rich data typing. Therefore, using XML Schemas instead of DTDs will be preferred for further development.

Implement the password protection

A model for password protection is already discussed in the "Distributed Database" section in chapter III. The following figure shows a more detailed access right model:

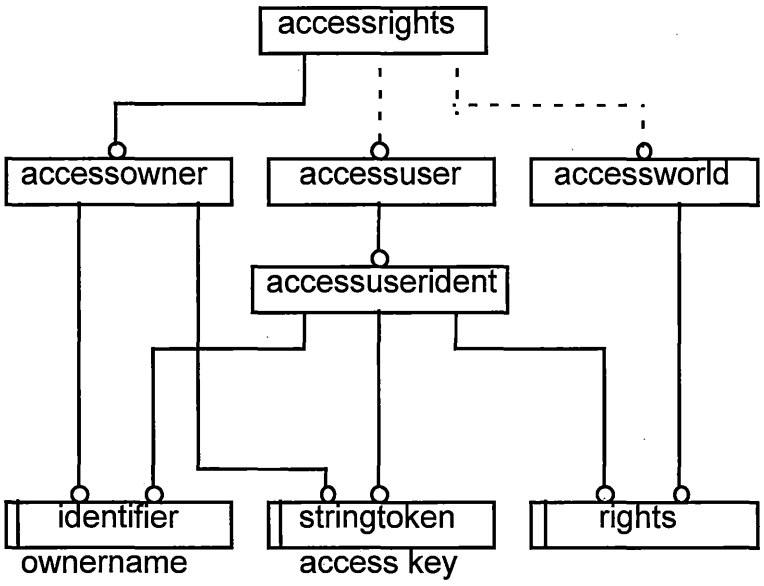

Figure 21 Access right model

Open an XML from a URL

It will be appropriate to replace the menus in CMT with buttons on an applet. Then clicking an "open" button should display a text box where a desired URL of an XML file may be entered.

Display Properties

Display properties are e.g. top left corner co-ordinates, size, scalability of a box, background color, box color, text location, text size, text style, font. The customer should be able to define all colors, but there should be only few core fonts: helvetica, courier & times.

Appendix B

The Source code

Following is the complete source code the Cost Modelling tool.

```
/*
 * CostModel.java                                    Last modified: 7/24/00
 *
 * Author: Mohammed Rahman
 *
 * Project: Cost Modelling Tool: User Interface and Edit Functons
 *
 * Description: This is a standalone java program that takes an XML file
 * as a command line argument. The XML file and all corresponding DTD
 * are then validated and parsed using SUN's SAX parser. This parsed object
 * is a DOM (Document Object Model) level 2 document. An AdapterNode class
 * is created to wrap the DOM nodes. The DomToTreeModelAdapter class
 * implements the TreeModel using the AdapterNode objects. Now a JTree is
 * is created using this TreeModel as its data model. An HTML pane is also
 * created to display corresponding values of tree node elements. Both the
 * JTree and HTML pane are then displayed to the user using a JSplitPane
 * GUI component. Any of the tree items excluding the root can be selected
 * and deleted using the "Delete" tree menu option. Once changes are made,
 * the new model can be saved as a new toXML.xml file by selecting "Save"
 * option from the file menu"
 */

import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.FactoryConfigurationError;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.DocumentBuilder;

import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;

import org.w3c.dom.Document;
import org.w3c.dom.DOMException;
```

```
// For write() operation
import com.sun.xml.tree.XmlDocument;
import java.io.*;
import java.io.File;
import java.io.IOException;

// Basic GUI components
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTree;

// Menu bar
import javax.swing.JMenuBar;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

// GUI components for right-hand side
import javax.swing.JSplitPane;
import javax.swing.JEditorPane;

// GUI support classes
import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.WindowEvent;
import java.awt.event.WindowAdapter;

// For creating borders
import javax.swing.border.EmptyBorder;
import javax.swing.border.BevelBorder;
import javax.swing.border.CompoundBorder;

// For creating a TreeModel
import javax.swing.tree.*;
import javax.swing.event.*;
import java.util.*;

public class CostModel extends JPanel
```

```
{
    // Global value so it can be ref'd by the tree-adapter
    static Document document;

       // Set up the tree
    protected JTree tree;
    protected TreeModel treeModel;

    // Possible to set this with a command-line argument
    boolean compress = true;
    static final int windowHeight = 460;
    static final int leftWidth = 300;
    static final int rightWidth = 340;
    static final int windowWidth = leftWidth + rightWidth;

    public CostModel()
    {
       // Make a nice border
       EmptyBorder eb = new EmptyBorder(5,5,5,5);
       BevelBorder bb = new BevelBorder(BevelBorder.LOWERED);
       CompoundBorder cb = new CompoundBorder(eb,bb);
       this.setBorder(new CompoundBorder(cb,eb));

       // Set up the tree
       treeModel = new DomToTreeModelAdapter();
       tree = new JTree(treeModel);
           tree.setEditable(false);

       // Need to iterate over the tree and make nodes visible.
       // Otherwise, the tree shows up fully collapsed
       //TreePath nodePath = ???;
       //tree.expandPath(nodePath);

           //Menu bar
           JMenuBar  menuBar = constructMenuBar();

       // Build left-side view
       JScrollPane treeView = new JScrollPane(tree);
       treeView.setPreferredSize(
           new Dimension( leftWidth, windowHeight ));

       // Build right-side view
```

```
    // must be final to be referenced in inner class
    final JEditorPane htmlPane = new JEditorPane("text/html","");
    htmlPane.setEditable(false);
    JScrollPane htmlView = new JScrollPane(htmlPane);
    htmlView.setPreferredSize(
        new Dimension( rightWidth, windowHeight ));

    // Wire the two views together. Use a selection listener
    // created with an anonymous inner-class adapter.
    tree.addTreeSelectionListener(
      new TreeSelectionListener() {
        public void valueChanged(TreeSelectionEvent e) {
          TreePath p = e.getNewLeadSelectionPath();
          if (p != null) {
            AdapterNode adpNode =
                (AdapterNode) p.getLastPathComponent();
            htmlPane.setText(adpNode.content());
          }
        }
      }
    );

    // Build split-pane view
    JSplitPane splitPane =
       new JSplitPane( JSplitPane.HORIZONTAL_SPLIT,
                       treeView,
                       htmlView );
    splitPane.setContinuousLayout( true );
    splitPane.setDividerLocation( leftWidth );
    splitPane.setPreferredSize(
         new Dimension( windowWidth + 10, windowHeight+10 ));

    // Add GUI components
    this.setLayout(new BorderLayout());
       this.add("North", menuBar );
    this.add("Center", splitPane );
} // constructor

public static void main (String argv [])
{
    if (argv.length != 1 ) {
        System.err.println ("Usage: java CostModel XmlFilename");
```

```
            System.exit (1);
        }

        DocumentBuilderFactory factory =
            DocumentBuilderFactory.newInstance();
        try {
            DocumentBuilder builder = factory.newDocumentBuilder();
            document = builder.parse( new File(argv[0]) );

            makeFrame();

        } catch (SAXParseException spe) {
            // Error generated by the parser
            System.out.println ("\n** Parsing error"
                + ", line " + spe.getLineNumber ()
                + ", uri " + spe.getSystemId ());
            System.out.println("   " + spe.getMessage() );

            // Use the contained exception, if any
            Exception  x = spe;
            if (spe.getException() != null)
                x = spe.getException();
            x.printStackTrace();

        } catch (SAXException sxe) {
            // Error generated by this application
            // (or a parser-initialization error)
            Exception  x = sxe;
            if (sxe.getException() != null)
                x = sxe.getException();
            x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
        }

    } // main
```

```
//Construct a menu
private JMenuBar constructMenuBar() {

            JMenu            menu;
            JMenuBar         menuBar = new JMenuBar();
            JMenuItem        menuItem;

            menu = new JMenu("File");
            menuBar.add(menu);

            menuItem = menu.add(new JMenuItem("Save"));
            menuItem.addActionListener(new SaveAction());

            menuItem = menu.add(new JMenuItem("Exit"));
            menuItem.addActionListener(new ActionListener() {
                    public void actionPerformed(ActionEvent e) {
                    System.exit(0);
                    }});

            /* Tree related stuff. */
            menu = new JMenu("Tree");
            menuBar.add(menu);

            menuItem = menu.add(new JMenuItem("Add"));
            //menuItem.addActionListener(new AddAction());

            menuItem = menu.add(new JMenuItem("Insert"));
            //menuItem.addActionListener(new InsertAction());

            menuItem = menu.add(new JMenuItem("Reload"));
            menuItem.addActionListener(new ReloadAction());

            menuItem = menu.add(new JMenuItem("Remove"));
            menuItem.addActionListener(new RemoveAction());

            return menuBar;
}

//Returns the TreeNode instance that is selected in the tree.
//If nothing is selected, null is returned.
   protected AdapterNode getSelectedNode() {
```

```
            TreePath selPath = tree.getSelectionPath();

            if(selPath != null)
                return (AdapterNode)selPath.getLastPathComponent();
            return null;
        }

        //SaveAction saves changes to the tree.
        class SaveAction extends Object implements ActionListener {
            //Saves the changes to the tree
            public void actionPerformed(ActionEvent e) {
                try {
                   XmlDocument xdoc = (XmlDocument) document;
                   FileOutputStream file = new FileOutputStream("toXml.xml");
                   //FileOutputStream file = new FileOutputStream(argv[1]);
                   xdoc.write (file);
                } catch (IOException ioe) {
                   // I/O error
                   ioe.printStackTrace();
                }

            }
    } //SaveAction


        //RemoveAction removes the selected node from the tree.  If
        //The root or nothing is selected nothing is removed.
        class RemoveAction extends Object implements ActionListener {
            // Removes the selected item as long as it isn't root.
            public void actionPerformed(ActionEvent e) {
                AdapterNode lastItem = getSelectedNode();

                if(lastItem != null && lastItem !=
(AdapterNode)treeModel.getRoot()) {
                    //treeModel.removeNodeFromParent(lastItem);
                    org.w3c.dom.Node parent = lastItem.domNode.getParentNode();
                    parent.removeChild(lastItem.domNode);

                    Object newRoot = null;
                    TreePath path = tree.getSelectionPath();
                    TreePath parentPath = null;
                    if (path != null) {
```

```
                                parentPath = path.getParentPath();
                                newRoot = parentPath.getLastPathComponent();
                        }
                        //TreeModelEvent(Object source, TreePath path, int[]
childIndices, Object[] children);
                        //TreeModelEvent evnt = new TreeModelEvent(treeModel, path,
new int[] {childIndex},new Object[] {newRoot});
                        //treeModel.valueForPathChanged(parentPath, newRoot);
                        //treeModel.valueForPathChanged(path, newRoot);
                        TreeModelEvent evnt = null;
                        evnt = new TreeModelEvent(this, parentPath);

    //((DomToTreeModelAdapter)treeModel).fireTreeNodesRemoved(evnt);

    ((DomToTreeModelAdapter)treeModel).fireTreeStructureChanged(evnt);
                }
            }
    } //RemoveAction


    class ReloadAction extends Object implements ActionListener
    {
            public void actionPerformed(ActionEvent e) {
                //How do I get path to the root?
                //AdapterNode rootNode = (AdapterNode)treeModel.getRoot();
                TreeModelEvent evnt = null;
                TreePath rootPath = tree.getSelectionPath();
                if (rootPath != null) {
                        evnt = new TreeModelEvent(this, rootPath);

    ((DomToTreeModelAdapter)treeModel).fireTreeStructureChanged(evnt);
                }
            }
    } // ReloadAction


    public static void makeFrame() {
     // Set up a GUI framework
     JFrame frame = new JFrame("Cost Model");
     frame.addWindowListener(
       new WindowAdapter() {
         public void windowClosing(WindowEvent e) {System.exit(0);}
       }
     );
```

```
    // Set up the tree, the views, and display it all
    final CostModel echoPanel =
       new CostModel();
    frame.getContentPane().add("Center", echoPanel );
    frame.pack();
    Dimension screenSize =
       Toolkit.getDefaultToolkit().getScreenSize();
    int w = windowWidth + 10;
    int h = windowHeight + 10;
    frame.setLocation(screenSize.width/3 - w/2,
                      screenSize.height/2 - h/2);
    frame.setSize(w, h);
    frame.setVisible(true);
} // makeFrame

// An array of names for DOM node-types
static final String[] typeName = {
    "none",
    "Element",
    "Attr",
    "Text",
    "CDATA",
    "EntityRef",
    "Entity",
    "ProcInstr",
    "Comment",
    "Document",
    "DocType",
    "DocFragment",
    "Notation",
};
static final int ELEMENT_TYPE =   1;
static final int ATTR_TYPE =      2;
static final int TEXT_TYPE =      3;
static final int CDATA_TYPE =     4;
static final int ENTITYREF_TYPE = 5;
static final int ENTITY_TYPE =    6;
static final int PROCINSTR_TYPE = 7;
static final int COMMENT_TYPE =   8;
static final int DOCUMENT_TYPE =  9;
static final int DOCTYPE_TYPE =  10;
static final int DOCFRAG_TYPE =  11;
```

```
static final int NOTATION_TYPE = 12;

// The list of elements to display in the tree
    // Would be nice to read from a DTD instead of hardcoding.
static String[] treeElementNames = {
            "slideshow",
            "slide",
            "title",          // For slideshow #1
            "slide-title",    // For slideshow #10
            "item",
            //"integertoken",
            "identifier",
            "realtoken",
            //"stringtoken",
            "accesskey",
            "accessowner",
            "accessuser",
            "accessuserident",
            "accessworld",
            "add",
            "author",
            "backgroundcolor",
            "bordercolor",
            "borderwidth",
            "cell",
            "cellcontents",
            "cellcost",
            "celldesigncost",
            "designcost",
            "dftcost",
            "celldisplay",
            "celldisposalcost",
            "cellencrypted",
            "cellmaintenancecost",
            "cellmanufacturingcost",
            "cellreadable",
            "cellref",
            "cellrefexternal",
            "celltype",
            "city",
            "color",
            "comment",
```

```
"company",
"country",
"dataorigin",
"department",
"differential",
"displayposition",
"documentation",
"div",
"exp",
"faculty",
"fp",
"globalvariable",
"increment",
"integral",
"log",
"maintenancecost",
"maintenancetestcost",
"manufacturingcost",
"minomaxvalue",
"modelhistory",
"modelhistoryelement",
"modelversion",
"mtbf",
"mttr",
"mult",
"neg",
"number",
"program",
"programname",
"reliability",
"reliabilityfunction",
"rights",
"root",
"scmheader",
"scmversion",
"scmXML",
"state",
"street",
"sub",
"system",
"term",
"testcost",
```

```
            "timestamp",
            "unitdefinition",
            "unit",
            "unitbase",
            "unitref",
            "value",
            "variable",
            "version",
            "written",
            "zipcode",
};

boolean treeElement(String elementName) {
  for (int i=0; i<treeElementNames.length; i++) {
            //System.out.println("Element name " + i + ":" +treeElementNames[i]);
            if ( elementName.equals(treeElementNames[i]) )
                    return true;
  }
  return false;
}

// This class wraps a DOM node and returns the text we want to
// display in the tree. It also returns children, index values,
// and child counts.
public class AdapterNode
{
  org.w3c.dom.Node domNode;

  // Construct an Adapter node from a DOM node
  public AdapterNode(org.w3c.dom.Node node) {
    domNode = node;
  }

  // Return a string that identifies this node in the tree
  public String toString() {
    String s = typeName[domNode.getNodeType()];
          if (s != "Document")
                  s = " ";
    String nodeName = domNode.getNodeName();
    if (! nodeName.startsWith("#")) {
       s += nodeName;
    }
```

```
    if (compress) {
       String t = content().trim();
       int x = t.indexOf("\n");
       if (x >= 0) t = t.substring(0, x);
       //s += " " + t;
       return s;
    }
    if (domNode.getNodeValue() != null) {
       if (s.startsWith("ProcInstr"))
          s += ", ";
       else
          s += ": ";
       // Trim the value to get rid of NL's at the front
       String t = domNode.getNodeValue().trim();
       int x = t.indexOf("\n");
       if (x >= 0) t = t.substring(0, x);
       s += t;
    }
    return s;
  } //toString

  public String content() {
    String s = "";
    org.w3c.dom.NodeList nodeList = domNode.getChildNodes();
    for (int i=0; i<nodeList.getLength(); i++) {
      org.w3c.dom.Node node = nodeList.item(i);
      int type = node.getNodeType();
      AdapterNode adpNode = new AdapterNode(node); //inefficient, but works

      if (type == ELEMENT_TYPE) {
                  // System.out.println("Element tag name = " + node.getNodeName());
        // Skip subelements that are displayed in the tree.
                    if (treeElement(node.getNodeName())) continue;

        s += "<" + node.getNodeName() + ">";
        s += adpNode.content();
        s += "</" + node.getNodeName() + ">";
      } else if (type == TEXT_TYPE) {
        s += node.getNodeValue();
      } else if (type == ENTITYREF_TYPE) {
        // The content is in the TEXT node under it
        s += adpNode.content();
```

```
    } else if (type == CDATA_TYPE) {
      // The "value" has the text, same as a text node.
      // Convert angle brackets and ampersands for display
      StringBuffer sb = new StringBuffer( node.getNodeValue() );
      for (int j=0; j<sb.length(); j++) {
        if (sb.charAt(j) == '<') {
          sb.setCharAt(j, '&');
          sb.insert(j+1, "lt;");
          j += 3;
        } else if (sb.charAt(j) == '&') {
          sb.setCharAt(j, '&');
          sb.insert(j+1, "amp;");
          j += 4;
        }
      }
      s += "<pre>" + sb + "\n</pre>";
    }
     // Ignoring these:
     //   ATTR_TYPE      -- not in the DOM tree, but can be accessed
           //                              -- through getAttribute() function call
     //   ENTITY_TYPE    -- does not appear in the DOM
     //   PROCINSTR_TYPE -- not "data"
     //   COMMENT_TYPE   -- not "data"
     //   DOCUMENT_TYPE  -- Root node only. No data to display.
     //   DOCTYPE_TYPE   -- Appears under the root only
     //   DOCFRAG_TYPE   -- equiv. to "document" for fragments
     //   NOTATION_TYPE  -- nothing but binary data in here
  }
  return s;
} //content

// Return children, index, and count values
public int index(AdapterNode child) {
  //System.err.println("Looking for index of " + child);
  int count = childCount();
  for (int i=0; i<count; i++) {
    AdapterNode n = this.child(i);
    if (child.domNode == n.domNode) return i;
  }
  return -1; // Should never get here.
}
```

```
  public AdapterNode child(int searchIndex) {
    //Note: JTree index is zero-based.
    org.w3c.dom.Node node =
         domNode.getChildNodes().item(searchIndex);
    if (compress) {
      // Return Nth displayable node
      int elementNodeIndex = 0;
      for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
        node = domNode.getChildNodes().item(i);
        if (node.getNodeType() == ELEMENT_TYPE
        && treeElement( node.getNodeName() )
        && elementNodeIndex++ == searchIndex) {
           break;
        }
      }
    }
    return new AdapterNode(node);
  }

  public int childCount() {
    if (!compress) {
      return domNode.getChildNodes().getLength();
    }
    int count = 0;
    for (int i=0; i<domNode.getChildNodes().getLength(); i++) {
       org.w3c.dom.Node node = domNode.getChildNodes().item(i);

       // Note: Have to check for proper type.
       // The DOCTYPE element also has the right name
       if (node.getNodeType() == ELEMENT_TYPE
       && treeElement( node.getNodeName() )) {
                     ++count;
       }
    }
    return count;
  }
} //AdapterNode

// This adapter converts the current Document (a DOM) into
// a JTree model.
public class DomToTreeModelAdapter implements TreeModel {
```

```
// Basic TreeModel operations
public Object getRoot() {
        //System.err.println("Returning root: " +document);
        return new AdapterNode(document);
}
public boolean isLeaf(Object aNode) {
        // JTree determines whether the icon shows up to the left.
        // Return true for any node with no children
        AdapterNode node = (AdapterNode) aNode;
        if (node.childCount() > 0) return false;
        return true;
}
public int getChildCount(Object parent) {
        AdapterNode node = (AdapterNode) parent;
        return node.childCount();
}
public Object getChild(Object parent, int index) {
        AdapterNode node = (AdapterNode) parent;
        return node.child(index);
}
public int getIndexOfChild(Object parent, Object child) {
        AdapterNode node = (AdapterNode) parent;
        return node.index((AdapterNode) child);
}
public void valueForPathChanged(TreePath path, Object newValue) {
        //Update the user object.
        AdapterNode node = (AdapterNode)path.getLastPathComponent();
        //SampleData sampleData = (SampleData)node.getUserObject();
        node.domNode.setNodeValue((String)newValue);
        //sampleData.setColor(Color.green);
        //nodeChanged(aNode);
}

 private Vector listenerList = new Vector();
 public void addTreeModelListener(TreeModelListener listener) {
        if ( listener != null
        && ! listenerList.contains( listener ) ) {
           listenerList.addElement( listener );
        }
  }
 public void removeTreeModelListener(TreeModelListener listener) {
        if ( listener != null ) {
```

```
                listenerList.removeElement( listener );
            }
        }
        public void fireTreeNodesChanged( TreeModelEvent e ) {
            System.err.println("fireTreeNodesChanged is called ");
            Enumeration listeners = listenerList.elements();
            while ( listeners.hasMoreElements() ) {
              TreeModelListener listener =
                    (TreeModelListener)listeners.nextElement();
              listener.treeNodesChanged( e );
            }
        }
        public void fireTreeNodesInserted( TreeModelEvent e ) {
            System.err.println("fireTreeNodesInserted is called ");
            Enumeration listeners = listenerList.elements();
            while ( listeners.hasMoreElements() ) {
               TreeModelListener listener =
                     (TreeModelListener)listeners.nextElement();
               listener.treeNodesInserted( e );
            }
        }
        public void fireTreeNodesRemoved( TreeModelEvent e ) {
            System.err.println("fireTreeNodesRemoved is called ");
            Enumeration listeners = listenerList.elements();
            while ( listeners.hasMoreElements() ) {
              TreeModelListener listener =
                    (TreeModelListener)listeners.nextElement();
              listener.treeNodesRemoved( e );
            }
        }
        public void fireTreeStructureChanged( TreeModelEvent e ) {
            System.err.println("fireTreeStructureChanged is called ");
            Enumeration listeners = listenerList.elements();
            while ( listeners.hasMoreElements() ) {
              TreeModelListener listener =
                    (TreeModelListener)listeners.nextElement();
              listener.treeStructureChanged( e );
            }
        }
    } //DomToTreeModelAdapter
} //CostModel
```

Appendix C

DTD Documentation

Following is a complete documentation for a DTD in a style equivalent to what is contained in the EDIF [24]Information models.

accesskey (stringtoken)

The access key is a string token. The content can be a simple password or a more complex automatically generated key. The semantic depends on the access and encryption method implemented. At the current state of the development a password is expected.

accessowner (identifier, accesskey)

The owner has all rights. For identification the owner name and the owner access key is required. The owner has by default the modify right.

accessrights (accessowner, accessuser?, accessworld?)

Access rights are grouped in the three categories: owner, user and world. An owner must be defined. The default for user and world is locked.

accessuser (accessuserident+)

Multiple users can be registered. For each user a separate identification is required. If there is an accessuser then there must be at least one entry.

accessuserident (identifier, accesskey, rights)

The users are identified by their individual key. If a key matches one of the keys the corresponding right are applied.

accessworld (rights)

There is no name or key required for the world, just the access right.

add (term, term)

Adds all the terms. Add (f, p) is equivalent to f + p.

author (stringtoken, stringtoken?)

The first string token identifies the author. The optional second string token provides the e-mail address of the author.

backgroundcolor (color)

Defines the background color of the box.

bordercolor (color)

This is the color of the border of a box.

borderwidth (integertoken)

The width is given in pixels.

cell (identifier, accessrights, (cellreadable | cellencrypted))

The cell is the core part of the hierarchy. The name has been borrowed from the IC design domain. All elements of the hierarchy are of this type, independent of what they describe.

The cell content can be either readable or encrypted.

cellcontents (celltype, celldisplay?, (cellref), cellcost?, reliability?, comment?)*

The cell type provides a rough characterisation and the cell display provides attributes for the display of the cell so that the visualisation stays the same after save and restore.

The hierarchy is built up by referring to other cells. The cellref represents the necessary links to the other cells. It is omitted for the leaf cells

The cellcost point to the different types of costs tat can be associated to a cell.

Reliability function can be added that are evaluated during the cost calculation.

A comment can be added to all cells.

cellcost ((celldesigncost | cellmanufacturingcost | cellmaintenancecost | celldisposalcost)+)

Costs associated with this entity. The entity can have three different cost types: design, manufacturing, maintenance and disposal. If there are cell cost, then there must be at least one cost function.

celldesigncost (designcost?, dftcost?)

The entity describes the cost of a cell. Costs can occur in the design process itself, but also costs occur that are relevant for testing the cell later on. These costs are called Design For Testability (dft) costs.

At least one of the two cost types must be present.

celldisplay ((displayposition | backgroundcolor | bordercolor | borderwidth)+)

If the element is displayed on the screen we should take care that is shows up at the same position each time. If the celldisplay is present, then at least one of entries must be present. Each of the components can occur at most one time.

celldisposalcost (identifier, term, comment?)

This entity describes the disposal cost for this entity. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

cellencrypted (cellcontents)

The cell contents can be either readable or encrypted. If the cell content is encrypted, the key will be used to decrypt the contents and use the cell in the way described in the access right section. This is required if the model must not be transferred in a readable form.

cellmaintenancecost (maintenancecost?, maintenancetestcost?)

The cellmaintenancecost covers the cost for the maintenance as well as the cost for testing. At least one of the two items must be present.

cellmanufacturingcost (manufacturingcost?, testcost?)

The manufacturing costs for a cell consist of course primarily of the production cost. Depending on the type of cell the costs for testing can have the same order of magnitude as the production costs. In these cases it is appropriate to distinguish between the two cost types. If no distinction is possible, the test costs will not be present.

At least one of the two cost items must be present.

cellreadable (cellcontents)

The cell contents can be either readable or encrypted. If the content is readable the access rights determine if the cell content is displayed and

evaluated or not. The contents of the cell is transferred in a readable format, although the software will respect the access rights.

cellref (identifier | cellrefexternal)

Another cell can be referenced by its name if it is local in this file. The identifier is the name of the cell.

If the cell is nor contained in this file, a reference to an external cell must be built up.

cellrefexternal (identifier, identifier, identifier?)

The reference to an external cell consists of three identifiers. Their meaning in the sequence of their occurrence is:

The name of the cell itself (see cellref).

The name of the file that contains the model where the cell is described.

The location of the file. The URL does not include the file name itself. This identifier is optional.

celltype (("mechanic " | "electric" | "electronic" | "pneumatic" | "hydraulic" | "undefined")+)

Type of the entity. At least one of the types below must be given. Multiple entries are possible.

A break system e.g. can have mechanic, hydraulic elements, electric and electronic elements. So if the cost tree is searched for a certain category the break system should appear in all of the listed types.

city ((stringtoken)+)

The city name can consist of one or more tokens.

color (integertoken, integertoken, integertoken)

The color is given in the 24 bit RGB model. The three numbers are red, green, blue.

comment ((stringtoken)+)

The comment consists of simple text in this version.

Later on it can be expanded to contain formatted text.

company ((stringtoken)+)

The first string token gives the company name. The following string tokens are used to identity the department and/or working group.

country ((stringtoken)+)

The country can consist of one or more string tokens.

dataorigin (company, (author | department | street | city | state | zipcode | country))*

The company is the first element in the dataorigin statement. The following entities are optional and can follow in any arbitrary order.

department ((stringtoken)+)

Contains the name of the department. It can consist of one or more tokens.

designcost (identifier, term, comment?)

This entity describes the pure design cost for this entity. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

dftcost (identifier, term, comment?)

This entity describes the costs spent for design for testability measures. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

differential (term, identifier)

Describe that the term is differentiated to the variable.

displayposition (integertoken, integertoken)

Defines the position of a displayable entity. It refers to the top left corner of the item. The reference point is the top left corner of the screen. The position is given in pixels.

documentation ((stringtoken)+)

For the time being it is unformatted text here. It will be extended to hold formatted text in the future.

div (term, term)

Divides the two terms. The first term is the dividend, the second term is the divisor.

exp (term, term)

Defines the exponential function. The first term is the base and the second term is the exponent.

faculty (term)

Defines the faculty function: term!

fp (realtoken, realtoken)

The first integer token the basis of the floating-point number, the second one is the exponent.

globalvariable (identifier, value?, unitref?, increment?)

A global variable must have a name so that it can be referenced in a term later on.

The value is optional.

The unitref is optional.

An increment can be defined if it is reasonable.

Global variables are available in the complete model.

identifier (#PCDATA)

An identifier gives an object a name. Objects that can be referenced must have a name.

increment (number)

Defines the step width a variable can be modified in.

integral (term, identifier)

The term has to be integrated over the variable. The variable is referenced via its name (identifier).

integertoken (#PCDATA)

Represents an integer token.

log (term)

Defines the decimal logarithm of the term.

maintenancecost (identifier, term, comment?)

This entity describes the maintenance cost for this entity. If test costs are identified separately they must be described in the maintenancetestcost. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

maintenancetestcost (identifier, term, comment?)

The maintenacetestcost gives that part of the cost that is related to testing the cell for identifying or

finding a possible failure. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

manufacturingcost (identifier, term, comment?)

This entity describes the costs for manufacturing the cell. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

minomaxvalue (number, number, number)

This gives the range (min + max) and the nominal value. It consists of three numbers.

modelhistory ((modelhistoryelement)+)

If we have a history, then it must contain at least one entry. The elements must not be in a specific order.

modelhistoryelement (modelversion, written)

The history refers to the versions of the model and the conditions how the model was written.

modelversion (integertoken, integertoken, integertoken)

The version of the model is given by a set of three integer token, the release, version, and subversion.

mtbf (identifier, number)

The identifier refers to a bas unit or a self defined unit.

mttr (identifier, number)

The identifier refers to a bas unit or a self defined unit.

mult (term, term)

Multiplies the two terms.

neg (term)

Gives the negative value of the term.

number (integertoken | realtoken | fp)

A number can be either an integer or a floating-point number. Both numbers are in a decimal system.

program (programname, version?)

This construct identifies the program that has generated the file. There can be at most one version.

programname (stringtoken)

Contains the name of the program.

realtoken (#PCDATA)

Represents a real number. It consists of one or more digits, followed by a '.', followed by one or more digits. The number can be proceeded by a '-' to indicate that it is negative.

reliability (mtbf?, mttr?, reliabilityfunction?)

This entity contains information about the reliability of the cell. There can be the well known values for the mean time between failure (MTBF), the mean time to repair (MTTR) or an arbitrary function describing the failure distribution,

reliabilityfunction (identifier, term)

The reliability function gets an identifier and a value (term).

rights ("locked" | "evaluate" | "read" | "modify")

One of the four access rights must be defined.

Locked means that this cell is not touched at all and remains invisible.

Evaluate means that the algorithm calculating the cost functions can access the local functions. This privilege is sufficient for all operations concerning cost optimisation.

Read means that the cell can be displayed, but not modified. It provides input to the user on how the model is constructed.

Modify gives full access to the cell. This is only required when the model itself is going to be modified.

root (term, (identifier | number))

This gives the root of the term. The identifier refers to a variable. The number is a simple value, e.g. is the square root is calculated.

scmheader (modelversion, written, (documentation | history | unitdefinition))*

The header information is contained here. At most one of each documentation, history or unitdefinition is allowed.

scmversion (integertoken, integertoken, integertoken)

The model version used when this file was written. Model versions are indicated by a set of three integer numbers.

SCMXML (identifier, scmheader, system, (comment))*

The first item in the SCMXML is the name of the file (identifier). The identifier uniquely identifies this file and is used whenever this file is referenced.

The scmheader contains the general information about the system that is described in this file, in particular version, owner and history.

The comment is optional.

state ((stringtoken)+)

The state name can consist of one or more string tokens.

street ((stringtoken)+)

Contains the street name. It consists of one or more tokens.

stringtoken (#PCDATA)

The string token is contains a simple contiguous text string.

sub (term, term)

Subtracts the second term from the first term. Sub (a, b) is equivalent to a - b.

system ((globalvariable), comment?, (cell | cellref))*

The system is the top-level entity of the model and the root of the system hierarchy. The global variables are defined here and a comment can be added as in many places. The definition of the global variables is followed by an optional comment.

The core part of the entity is the top level cell, that can either be located here or referrred to using the cell reference mechanism.

term (number | minomax | add | differential | div | exp | faculty | integral | log | mult | neg | root | sub | identifier)

A term is the general expression of a mathematical object. It can be a number, one of the functions in this list or a reference to a variable. The variable

can be either a global variable or a local variable, representing a cost function.

testcost (identifier, term, comment?)

This entity describes costs bound to the testing of the cell. It requires an identifier for being able to refer to this particular cost function later on.

The term defines the cost function, and the comment is optional.

timestamp (integertoken, integertoken, integertoken, integertoken, integertoken, integertoken)

The time stamp contains a set of six integer numbers. They have the following meaning:

year, month, day, hour, minute, and second.

unitdefinition (unit+)

The unitdefinition contains a set of unit statements.

unit (identifier, unitbase, number)

Any units can be defined based on the basic units in the list. Millimeters would be e.g. defined as

unit (millimeter, 0.001, meter)

unitbase (ampere | candela | celsius | coulomb | degree | henry | hertz | joule | kelvin | meter | mole | ohm | radian | second | volt | weber)

This is the list of base units that are supported.

unitref (identifier)

Refers to one of the base units (refer to the list in *unit*) or one of the units defined in the header.

value (number | minomaxvalue)

A value can be either a simple number or a minomax value.

variable (identifier, unitref?, increment?)

A variable must have a name. The reference to a unit is optional, as well as the possible increment value. Variables are created for each cost function. The value for such a variable comes from the corresponding cost function.

version (stringtoken)

A text string containing the version number in whatever format is provided.

written (timestamp, dataorigin, program, comment?)

The written construct contains always a time stamp. The other parameters are optional.

zipcode (stringtoken)

The zip code is a number most of the time, but there are also countries with a mix of characters and digits.

Appendix D

DTD Example

This section of the appendix contains a DTD based on the definitions described in the previous section.

```
ScmXML.dtd

<!-- DTD for describing the Siegen Cost Model  -->

 <!-- ========================================================== -->

 <!-- The integertoken contains a signed integer value  -->

 <!ELEMENT integertoken (#PCDATA)>

 <!-- This defines a name or refers to a name of an element   -->

 <!ELEMENT identifier (#PCDATA)>

 <!--  Real numbers are defined as an optional sign, followed by a contiguous sequence of
caracters. Sign and value can be separted by white space. The character sequence consists
of 1..n digits, a decimal point, followed by 0..m digits. -->

 <!ELEMENT realtoken (#PCDATA)>

 <!-- The string token is one contiguous string -->

 <!ELEMENT stringtoken (#PCDATA)>

 <!-- ========================================================== -->

<!ELEMENT accesskey (stringtoken)>

<!ELEMENT accessowner (identifier, accesskey)>

<!ELEMENT accessuser (accessuserident+)>

<!ELEMENT accessuserident (identifier, accesskey, rights)>

<!ELEMENT accessworld (rights)>

<!ELEMENT add (term, term)>

<!ELEMENT author (stringtoken, stringtoken?)>
```

```
<!ELEMENT backgroundcolor (color)>

<!ELEMENT bordercolor (color)>

<!ELEMENT borderwidth (integertoken)>

<!ELEMENT cell (identifier, accessrights, (cellreadable | cellencrypted) )>

<!ELEMENT cellcontents (celltype, celldisplay?, (cellref)*, cellcost?, reliability?,
comment? )>

<!ELEMENT cellcost ((celldesigncost | cellmanufacturingcost | cellmaintenancecost |
celldisposalcost)+ )>

<!ELEMENT celldesigncost (designcost?, dftcost?)>

<!ELEMENT designcost (identifier, term, comment?)>

<!ELEMENT dftcost (identifier, term, comment?)>

<!ELEMENT celldisplay ((displayposition | backgroundcolor | bordercolor | borderwidth)+ )>

<!ELEMENT celldisposalcost (identifier, term, comment?)>

<!ELEMENT cellencrypted (cellcontents)>

<!ELEMENT cellmaintenancecost (maintenancecost?, maintenancetestcost?)>

<!ELEMENT cellmanufacturingcost (manufacturingcost?, testcost?)>

<!ELEMENT cellreadable (cellcontents)>

<!ELEMENT cellref (identifier | cellrefexternal)>

<!ELEMENT cellrefexternal (identifier, identifier, identifier?)>

<!ELEMENT celltype ((mechanic | electric | electronic | pneumatic | hydraulic |
undefined)+ )>

<!ELEMENT city ((stringtoken)+ )>

<!ELEMENT color (integertoken, integertoken, integertoken)>

<!ELEMENT comment ((stringtoken)+ )>

<!ELEMENT company ((stringtoken)+ )>

<!ELEMENT country  ((stringtoken)+ )>

<!ELEMENT dataorigin (company, (author | department | street | city | state | zipcode |
country)* )>

<!ELEMENT department ((stringtoken)+ )>

<!ELEMENT differential (term, identifier)>

<!ELEMENT displayposition (integertoken, integertoken)>

<!ELEMENT documentation ((stringtoken)+ )>
```

```
<!ELEMENT div (term, term)>

<!ELEMENT exp (term, term)>

<!ELEMENT faculty (term)>

<!ELEMENT fp (realtoken, realtoken)>

<!ELEMENT globalvariable (identifier, value?, unitref?,  increment?)>

<!ELEMENT increment (number)>

<!ELEMENT integral (term, identifier)>

<!ELEMENT log (term)>

<!ELEMENT maintenancecost (identifier, term, comment?)>

<!ELEMENT maintenancetestcost (identifier, term, comment?)>

<!ELEMENT manufacturingcost (identifier, term, comment?)>

<!ELEMENT minomaxvalue (number, number, number)>

<!ELEMENT modelhistory ((modelhistoryelement)+ )>

<!ELEMENT modelhistoryelement (modelversion, written)>

<!ELEMENT modelversion (integertoken, integertoken, integertoken)>

<!ELEMENT mtbf (identifier, number)>

<!ELEMENT mttr (identifier, number)>

<!ELEMENT mult (term, term)>

<!ELEMENT neg (term)>

<!ELEMENT number (integertoken | realtoken | fp)>

<!ELEMENT program (programname, version? )>

<!ELEMENT programname (stringtoken)>

<!ELEMENT reliability (mtbf?, mttr?, reliabilityfunction?)>

<!ELEMENT reliabilityfunction (identifier, term)>

<!ELEMENT rights (locked | evaluate | read | modify)>

<!ELEMENT root (term, (identifier | number) )>

<!ELEMENT scmheader (modelversion, written, (documentation | history | unitdefinition)* )>

<!--ELEMENT scmversion (0, 0, 7)-->

<!ELEMENT scmMXML (identifier, scmheader, system, (comment)* )>

<!ELEMENT state ((stringtoken)+ )>

<!ELEMENT street ((stringtoken)+ )>
```

```
<!ELEMENT sub (term, term)>

<!ELEMENT system ((globalvariable)*, comment?, (cell | cellref) )>

<!ELEMENT term (number | minomax | add | differential | div | exp | faculty | integral |
log | mult | neg | root | sub | identifier)>

<!ELEMENT testcost (identifier, term, comment?)>

<!ELEMENT timestamp (integertoken, integertoken, integertoken, integertoken, integertoken,
integertoken)>

<!ELEMENT unitdefinition (unit+)>

<!ELEMENT unit (namedef, unitbase, number)>

<!ELEMENT unitbase (ampere | candela | celsius | coulomb | degree | henry | hertz | joule
| kelvin | meter | mole | ohm | radian | second | volt | weber)>

<!ELEMENT unitref (identifier)>

<!ELEMENT value (number | minomaxvalue)>

<!ELEMENT variable (identifier, unitref?, increment?)>

<!ELEMENT version (stringtoken)>

<!ELEMENT written (timestamp, dataorigin, program, comment? )>

<!ELEMENT zipcode (stringtoken)>
```

Appendix E

Example: XML Data File

The example consists of two files. The first one, contained in this section, gives the framework for the complete date set. All global definitions are contained her, and a reference to the top level cost bearing entity.

Example.xml

```
<?xml version="1.0" encoding='UTF-8'?>
<!DOCTYPE scmXML SYSTEM "scmXML.dtd">
<scmXML>
  <identifier> example_01 </identifier>
    <scmheader>
      <modelversion>
        <integertoken> 0 </integertoken>
        <integertoken> 0 </integertoken>
        <integertoken> 1 </integertoken>
       </modelversion>
       <written>
         <timestamp>
           <integertoken> 2000 </integertoken>
           <integertoken> 06 </integertoken>
           <integertoken> 20 </integertoken>
           <integertoken> 17 </integertoken>
           <integertoken> 00 </integertoken>
           <integertoken> 00 </integertoken>
         </timestamp>
       <dataorigin>
         <company> Universitaet Siegen </company>
         <author> Michael G. Wahl </author>
         <country> Germany </country>
       </dataorigin>
```

```
      <program>
        <programname> manually </programname>
      </program>
    </written>
    <unitdefinition>
      <unit>
        <identifier> year </identifier>
        <unitbase> second </unitbase>
        <number> 31536000 </number>
      </unit>
    </unitdefinition>
  </scmheader>
<system>
  <globalvariable>
    <identifier> manufactured_units </identifier>
    <value>
      <number> 200000 </number>
    </value>
  </globalvariable>
  <globalvariable>
    <identifier> manufacturing_periode </identifier>
    <value>
      <number> 5 </number>
    </value>
    <unitref>
       <identifier> year </identifier>
    </unitref>
  </globalvariable>
  <globalvariable>
    <identifier> product_life_time </identifier>
    <value>
      <number>15 </number>
    </value>
    <unitref>
       <identifier> year </identifier>
    </unitref>
  </globalvariable>
  <comment> External cell reference. </comment>
  <cellref>
    <cellrefexternal>
      <identifier> automobile </identifier>
      <identifier> Example_Cell_File.xml </identifier>
```

```
      </cellrefexternal>
    </cellref>
  </system>
  <comment> This is the first approach. </comment>
</scmXML>
```

Appendix F

Example: File Describing The Cells

This section contains the cells that go into the framework provided by the main file of the previous section.

Example_Cell_File.xml

```
<allcells>
<cell>
  <identifier> automobile </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
```

```
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic electric electronic hydraulic
      </celltype>
      <cellref>
        <identifier> body </identifier>
      </cellref>
      <cellref>
        <identifier> drive </identifier>
      </cellref>
      <cellref>
        <identifier> power_electric </identifier>
      </cellref>
      <cellref>
        <identifier> electronic </identifier>
      </cellref>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> body </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
```

```
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
      <cellcost>
        <cellmanufacturingcost>
          <identifier> body_manufacturing </identifier>
          <term>
            <number>
              <integertoken> 5000 </integertoken>
            </number>
          </term>
        </cellmanufacturingcost>
      </cellcost>
      <reliability>
      </reliability>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> drive </identifier>
  <accessrights>
    <accessowner>
```

```
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
      <cellref>
        <identifier> engine </identifier>
      </cellref>
      <cellref>
        <identifier> gearbox </identifier>
      </cellref>
      <cellref>
        <identifier> drive_train </identifier>
      </cellref>
      <cellref>
        <identifier> wheel_assembly </identifier>
      </cellref>
```

```
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> power_electric </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> electric
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> electronic </identifier>
```

```
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> electronic
      </celltype>
      <cellref>
        <identifier> dashboard </identifier>
      </cellref>
      <cellref>
        <identifier> motor_control </identifier>
      </cellref>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
```

```
<identifier> dashboard </identifier>
<accessrights>
  <accessowner>
    <identifier> Michael_G_Wahl </identifier>
    <accesskey>
      <stringtoken> michael </stringtoken>
    </accesskey>
  </accessowner>
  <accessuser>
    <accessuserident>
      <identifier> mohammed </identifier>
      <accesskey>
        <stringtoken> rahman </stringtoken>
      </accesskey>
      <rights> modify </rights>
    </accessuserident>
  </accessuser>
</accessrights>
<cellreadable>
  <cellcontents>
    <celltype>
    </celltype>
    <cellref>
      <identifier> processor </identifier>
    </cellref>
    <cellref>
      <identifier> periphery </identifier>
    </cellref>
    <cellref>
      <identifier> board </identifier>
    </cellref>
```

```
        <cellref>
          <identifier> edge_connectors </identifier>
        </cellref>
      </cellcontents>
    </cellreadable>
  </cell>
  <cell>
    <identifier> motor_control </identifier>
    <accessrights>
      <accessowner>
        <identifier> Michael_G_Wahl </identifier>
        <accesskey>
          <stringtoken> michael </stringtoken>
        </accesskey>
      </accessowner>
      <accessuser>
        <accessuserident>
          <identifier> mohammed </identifier>
          <accesskey>
            <stringtoken> rahman </stringtoken>
          </accesskey>
          <rights> modify </rights>
        </accessuserident>
      </accessuser>
    </accessrights>
    <cellreadable>
      <cellcontents>
        <celltype> electronic
        </celltype>
      </cellcontents>
    </cellreadable>
```

```
</cell>
<cell>
  <identifier> engine </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic electronic
      </celltype>
      <cellref>
        <identifier> motor_control </identifier>
      </cellref>
      <cellref>
        <identifier> motor </identifier>
      </cellref>
    </cellcontents>
```

```
  </cellreadable>
</cell>
<cell>
  <identifier> motor_control </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> electronic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> motor </identifier>
  <accessrights>
```

```
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> gearbox </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
```

```
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> drive_train </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
```

```
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> wheel_assembly </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
```

```
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype>
      </celltype>
      <cellref>
        <identifier> rubber_bearing </identifier>
      </cellref>
      <cellref>
        <identifier> wheel_break </identifier>
      </cellref>
      <cellref>
        <identifier> wheel_bearing </identifier>
      </cellref>
      <cellref>
        <identifier> shocks </identifier>
      </cellref>
      <cellcost>
      </cellcost>
      <reliability>
      </reliability>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> rubber_bearing </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
```

```
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> wheel_break </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
```

```
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic electric
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> wheel_bearing </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
```

```
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
<cell>
  <identifier> shock_absorber </identifier>
  <accessrights>
    <accessowner>
      <identifier> Michael_G_Wahl </identifier>
      <accesskey>
        <stringtoken> michael </stringtoken>
      </accesskey>
    </accessowner>
    <accessuser>
      <accessuserident>
        <identifier> mohammed </identifier>
        <accesskey>
          <stringtoken> rahman </stringtoken>
        </accesskey>
        <rights> modify </rights>
      </accessuserident>
    </accessuser>
  </accessrights>
  <cellreadable>
    <cellcontents>
      <celltype> mechanic
      </celltype>
    </cellcontents>
  </cellreadable>
</cell>
</allcells>
```

REFERENCES

[1] "Applet Security", Sun Microsystems, 2000, http://java.sun.com/sfaq/

[2] Brooks, F., "The Mythical Man-Month : Essays on Software Engineering," Addison-Wesley, 1995

[3] Dear, I.; Dislis, C.; Dick, J.; Ambler, A.: "Test Strategy Planning Using Economic Analysis." In: Economics of Electronic Design, Manufacture and Test, M. & A. Ambler (Eds.); Kluwer Academic Publishers, 1994

[4] EIA; INCONSE; EPIC; "EIA/IS 731.1-Systems Engineering Capability Model", 1998

[5] "EIA 632 Processes for Engineering a System." Electronic Industry Alliance, 1999

[6] "EIA 731.1 System Engineering Capability Model." Electronic Industry Alliance, 1999

[7] "European Vanguard Effort on Research of Systems for Testing". Project of the Commission of the European Community. 1989-1993

[8] "EXPRESS Language Reference Manual"; ISO TC 184/SC 4/WG 5, Document N14; 1991

[9] "Extensible Markup Language (XML) 1.0". W3C Recommendation, 1998

[10] Fabricky, W.J.; Blanchard, B.S.: "Life-Cycle Costs and Economic Analysis." Prentice Hall, 1991.

[11] Farren, Des; Ambler, A.: "System Test Cost Modelling based on Event Rate Analysis." Proceedings of the 3rd Intl. Conference on Economics of Design, Test & Manufacturing, Austin, 1994, IEEE CS Press 1996.

[12] Ford, N.; Weber, E.; Azzouka, T.; Dietzler, T.; Streeter, J.; Williams, C. "Borland JBuilder 3 Unleashed," Sams Publishing, USA, 1999

[13] "Java Project X Technology Release 1," Sun Microsystems, 2000, http://java.sun.com/features/1999/03/xml-side1.html,

[14] Kaminski, J. "Concept for a Cost Modelling Tool," Diploma Thesis, 1999

[15] Kececioglu, D.: "Maintainability, Availability & Operational Readiness Engineering". Prentice Hall, 1995

[16] Lemay, L; Cadenhead, R. "Teach Yourself Java 2 in 21 Days," Sams Publishing, USA, 1999

[17] Morrison, et al., "XML Unleashed – From Knowledge to Mastery," Sams Publishing, USA, 2000

[18] Piroumian, V. "Java GUI Development – The Authoritative Solution," Sams Publishing, Indianapolis, Indiana, 1999

[19] Riedel, T.; "Effective Testability Design for the Product's Life-Cycle," Diploma Thesis; 1999

[20] Riedel, T.; Wahl, M.; Ambler, A.; "Cost Analysis linking Deign, Test & Maintenance". European Test Workshop, Constance, 1999

[21] Riedel, T.; Wahl, M.: Ambler, A.: "Design for Testability and Maintenance for Long Lasting Systems". WDTA, Dubrovnik, 1999

[22] Riedel, T.; Ambler, A.; Wahl, M.; "LCCA: Life Cycle Cost Analysis". AUTOTESTCON, Salt Lake City, 1998

[23] W3C: "XML Schema". Various documents on XLS schema definitions, www.w3c.org, 2000

[24] Wahl, M. (pub.): EDIF Test Adjunct Standard Version 1 to EDIF Version 3~0~0, Part 1: Information Model, Part 2: Reference. Electronics Industries Association, Washington, D.C., 1993

[25] Wei, S.; Nag, P.K.; Blanton, R.D.; Gattiker, A.; Maly, W.: "To DFT or not to DFT?" IEEE ITC 1997, Washington, 1997, pp. 557 - 566

[26] "A Quick Introduction to XML," Sun Microsystems, 2000, http://java.sun.com/xml/docs/tutorial/overview/1_xml.html

VITA

Mohammed Rahman was born in Jessore, Bangladesh, on January 9, 1972, the son of Atiur Rahman and Monowara Khanam. He completed his high school in Jessore, Bangladesh, and received his Bachelor of Science in Electrical Engineering in December 1995 from the University of Texas at Austin. Since July 1996, he has been working at VTEL Corporation as a Software Engineer. He is currently pursuing his Master's degree in Software engineering at the Southwest Texas State University, San Marcos, Texas.

Permanent Address: 2611 Bee Cave Road #209

Austin, Texas 78746

This thesis was typed by Mohammed Rahman.