AN OBJECT-ORIENTED COMPONENT-BASED APPROACH TO BUILDING REAL-TIME SOFTWARE SYSTEMS

André Baas

A project report submitted to the Faculty of Engineering, University of Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg 1993

DECLARATION

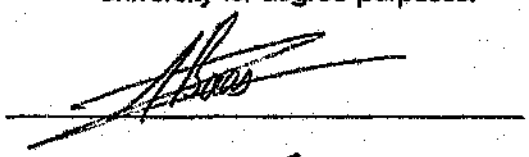
This Project Report is being submitted for the Degree of Master of Science in Engineering in the University of Witwatersrand, Johannesburg.

I declare that:

a) This Project Report is largely my own unaided work. I have been instrumental in initiating, guiding and managing a programme at my employers, BSW-Data (Pty) Ltd, on which my research for the Project Report is based. Because the programme at BSW-Data will eventually be far more extensive, the Project Report is contained to the specific area of interest as described in the Abstract.

I have received assistance from my employer in terms of facilities, literature, and real-time systems design expertise. I did receive input from project team members on detailed technical issues, related to the object-oriented software development project.

b) This Project Report, or any part thereof, has not been previously submitted to any university for degree purposes.



28TH day of FEBRUARY 1993

ABSTRACT

This Project Report reports on the study of an approach to building integrated real-time software systems based on re-usable object-oriented components. The basis of the approach is the development of a 3-layered structure of components, where each layer is built on the underlying layer of components.

The lower layer of components consists of generic re-usable building blocks that may be re-used for building and integrating other real-time applications. The middle layer consists of components that are generic to the application domain, and the top layer consists of components that are specific to each application of that application domain.

The Report includes researching and developing methods of communicating between these building blocks using an OSI/CMIP-conformant 'software highway', and in this regard particular attention is given to the formal and de facto industry standards.

With this approach, it is argued that the application engineer can effectively build new applications using the re-usable components. This is demonstrated by reporting on the implementation of a large real-world Telecommunications Network Management application.

The Project Report contains a critical analysis of the technical, organisational and project management issues of this object-oriented component approach as compared to the traditional development approach. The Report concludes that despite certain technical and organisational concerns, the object-oriented approach does indeed yield several worthwhile benefits for developing real-time software systems. These benefits include genuine re-usability, and improved productivity, testability and maintainability.

DEDICATION

To fulfilled dreams, and to my wife Ria.

ACKNOWLEDGEMENTS

To the '*AccessView*' project team at BSW-Data for their invaluable input, and to BSW-Data for the facilities, encouragement and support.

LIST OF ABBREVIATIONS AND DEFINITIONS

ABBR	MEANING
AccessView	A proprietary market name for the core components plus a subset of application components required for telecommunications network management applications.
ADT	Abstract Data Type.
ANSI	American National Standards Institute.
API	Application Programme Interface
BER	Bit Error Rate.
Building Blocks	Re-usable generic components from *ObjectView* and *AccessView* eg. RTOMS, WMI, MMI, and subsets thereof.
CAE	Common Application Environment.
CASE	Computer Aided Software Engineering.
CMIP	Common Management Information Protocol.
CMIS	Common Management Interface Services.
CMISE	Common Management Interface Service Element.
CMOQ	Common Management Interface over QCOMM.
Components	Another term for building blocks.
Core components	The basic MMI, WMI, RTOMS, CONFIGURATOR and HISTORIAN components.
CUI	Character User Interface.
CustomView	A proprietary name for the customised portion of applications.
DBOM	Database Object Manager.
DSM	Database Services Manager.
ETSI	European Telecommunications Standards Institute.
Framework	The term used by recognised authors for a collection or abstraction of object/class components that are specific to an application, and provide reuse at the largest granularity
GOSIP	Government OSI Profile programme (USA).
GUI	Graphical User Interface.
HOM	Historian Object Manager.
LabView	A proprietary market name for the core components plus a subset of application components required for laboratory applications.
MMI	Man-machine Interface.
MO	Managed Object.
NE	Network Element.
ObjectView	The proprietary market name for the toolbox of generic components suited for re-use in a variety of technical real-time applications. (Note that this is a proprietary name which originated from the early days of this project, and should not be confused with Borland's *ObjectView* product).
ODBMS	Object Database Management System.
OM	Object Manager.
OOA	Object Oriented Analysis.
OOD	Object Oriented Design.
OOP	Object Oriented Programming.

ABBR	MEANING
ProcessView	A proprietary market name for the core components plus a subset of application components required for supervisory monitoring and control applications.
QCOMM	Proprietary process-to-process communications subsystem.
QLINK	Proprietary cross-nodal communications subsystem using TCP/IP.
RTOMS	Real-time Object Management System.
SNMP	Simple Network Management Protocol.
SOMI	BSW-Data's Standard Object Management Interface.
TNM	Telecommunications Network Management.
Toolkit/Toolbox	Refers to a repository containing building blocks or components. (Although this is the widely used meaning, and is used throughout this report, it is technically not accurate. A toolbox/toolkit implies 'tools' to aid the developer to design, build and test systems. However in this context it contains components from which new systems are built).
WMI	World Machine Interface.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 STATEMENT OF PROBLEM

Of all the monsters that fill the nightmares of our folklore, none terrify more than werewolves, because they transform unexpectedly from the familiar into horrors. For these, one seeks bullets of silver that can magically lay them to rest. The familiar software project, at least as seen by the non-technical manager, has something of this character; it is usually innocent and straightforward, but is capable of becoming a monster of missed schedules, blown budgets, and flawed products. So we hear desperate cries for a silver bullet - something to make software costs drop as rapidly as computer hardware costs do.

- Brooks[15].

The above quote from Brooks[15] classically summarises the problems of many software projects.

The traditional functional decomposition and structured analysis approaches to building custom software projects has many problems. These problems include poor productivity, low re-usability and complexity of maintenance and testing.

As students we have been taught all the good things about the traditional structured analysis and design methodology[21,25,33]. As developers, we have all enjoyed starting each new project with a clean sheet of paper, relishing the challenges of designing and writing software systems from scratch.

However, it is only a matter of time and experience before the fundamental problems are appreciated. Initial excitement and commitment gives way to looming deadlines and frantic activity. Problems of scope and complexity under-estimation, re-designs and software changes, and the resulting overruns of budgets and time-scales. And after the dreaded tail-end the post-mortem follows.

In recent years I have gained better appreciation of the problems, especially as seen from the view of managing software projects. At times it appeared that development teams were not practising the structured methodologies properly - so additional effort was placed on further improving skills, and utilising CASE (Computer Aided Software Engineering) technologies in the development process.

Apart from improving skills, two factors were evident in applying the traditional structured analysis methodology to custom systems development:

Reuse - Although development teams did re-use software libraries to a very limited extent, there was no substantial software re-use taking place,

Maintainability - Implementing software changes both during and after development was complex and costly.

The above issues and others have been well researched and are claimed to be the primary causes of poor productivity and quality in software systems. For these reasons there has been a strong call for software re-usability and the use of re-usable software components implemented with the object-oriented paradigm. Some of the noted authors who have done research in these fields include Brooks [15], Cox[20,19], Yourdon[58], Lewis et al[41], Meyer[46,44], Gibbs et al[27], Helm and Maarek[34], Barnes and Bollinger[5,27], and others.

With the advent of the object-orientation paradigm in recent years, there has been a serious challenge to the traditional paradigms. The reason is that research has confirmed that the object-oriented paradigm is particularly suited to building re-usable components. Interesting studies on this have been done by Booch[13], Brooks[15], Cox[20] and Yourdon[58].

1.2 CLARIFICATION OF TERMS

1.2.1 Definition of the Project Report

This Project Report is submitted in partial fulfilment of the requirements for the Degree of Master of Science in Engineering. It is based on a large software development project currently being funded and resourced by my employers, BSW-Data.

It is important to note that this software development project was not initiated as part of the Project Report, but it was believed that the subject of the Project Report would greatly benefit the development project, and likewise the development project would benefit the Project Report.

For this reason it is necessary to clearly differentiate between the software development project, and this Project Report. Throughout this text, the development project is referred to as the 'project', and the Project Report will be referred to as the 'report' or the 'study'.

1.2.2 Definition of Real-time Systems

For the purposes of this report, it is necessary to define what is meant by real-time systems, seen in context of our current application domain. The Telecommunications Network Management (TNM) application domain is the focus of this study. This application domain is real-time in terms of the timing, asynchronous and distributed processing of real-time information, requiring real-time responses from seconds to milliseconds. References to 'real-time' in this report does not extend to sub-microsecond mission-critical applications. (Refer to Section 1.8 of this Chapter for a review of the telecommunications network management domain).

1.2.3 Definition of Technical Systems

In this report, reference to 'technical systems' is made. This is indeed a very broad reference. For the purposes of this study the term 'technical systems' will be defined as non-commercial software systems that are used to monitor, process, report on, and perform control on external real-world physical objects. Good examples are systems for process-control supervisory systems, telecommunications network management applications, electricity reticulation, laboratory control etc.

1.3 OBJECTIVE OF STUDY

The objective of this study was to investigate and develop an approach to building real-time software systems based on re-usable object-oriented components. A further objective was to assess whether this approach was successful, particularly in terms of improved re-usability, maintainability and productivity.

In terms of the above global objective, there were five major focuses in this study:

(1) To research, manage and guide the design and development of an object-oriented component approach to building real-time software systems. The building blocks, or software I.C.s (Cox[20,19]), were to be implemented using the object-oriented methodology and technologies. The goal was to create a 'toolbox' of generic, re-usable building blocks that could be re-used to build and integrate new distributed real-time applications.

(2) The second major focus was to guide the design and development of suitable and practical component structures and layers for the various subsystems so as to further augment the re-usability.

(3) The third focus was to research and develop methods of communicating between these building blocks along a 'software highway'. Particular attention was to be given to the use of object-oriented forms of industry standard protocols, such as OSI's CMIP/CMIS protocol and services, implemented on the OSI, TCP/IP and proprietary communications stacks.

(4) The fourth focus was to use these components to build a real-world telecommunications network management application.

(5) The final focus was to critically analyse the issues involved in adopting the object-oriented approach to building real-time systems.

With this 'toolbox' of generic core components, the objective is for the application programmer or 'domain expert' to rapidly build new applications, where it would be necessary to develop only that portion of the system pertaining to the specific application domain eg. network management, materials handling, water treatment, plant control.

The research includes a report on implementation of a real Telecommunications Network Management application based on the 'toolbox' of generic object-oriented software components. Examples of design approaches employed and notation methods to document generic classes and application object classes are also included.

The technical and project management/control issues and metrics have been monitored and recorded during implementation of the application. The project report concludes with a critical analysis of these issues as compared to the custom software development using the traditional structured analysis paradigm.

1.4 WHAT IS NEW IN THIS STUDY

There is an abundance of literature on object-orientation written by noted authors such as Booch, Meyer, Cox, Yourdon, Lewis, Wirfs-Brock and others (see references). Much of the literature covers topics such as object-oriented analysis, design and programming, as well as re-use and design and management of re-usable class libraries. Further research and literature exists on object-oriented databases, project life-cycles and project management.

However, most of these studies are in-depth research of particular topics. There appears to be limited consolidated work which addresses all the issues concerning the *practical application* of this paradigm in the development of real-time technical systems. (Refer to section 1.2 of this chapter for the definition of technical real-time systems). These issues range from the technical to broader organisational issues. It is these issues that have been studied, researched and reported in this research report.

There is however, one very interesting study done by Ahmad et al[1] on the application of object-oriented software in real-time systems. Here the two vastly different paradigms are used by rate project teams to develop the same functional system. The findings of Ahmad's research are frequently referred to, corroborated and challenged in this study.

1.5 SCOPE OF THIS WORK

The scope of the project report is to research and guide the development of an object-oriented approach to building generic building blocks that can be practically and effectively used for applications in Telecommunications Network Management.

The topic has personally been a subject of intense interest since early 1991, and as such I have been involved in the recent initiation of the development project.

I am performing the project management functions on the development project, and the development team consists of an average of 14 systems engineers. In this role I have had close involvement in the design and project management aspects of the project, as well as being responsible for the management, control and direction of the re-usability programme in the organisation.

This project report is not intended to be a computer science study of object-orientation as applied to building re-usable components. Instead it is software engineering research and studies a component approach to building real-time systems using the object-oriented paradigm.

To clarify the definition of software engineering, Macro and Buxton[43] define software engineering as *'... a whole set of activities needed to produce high quality software systems within known limitations of resources such as time, effort, money and equipment. These activities include specifications and feasibility studies, programming, quality control and assurance, and documentation.'*

However, this definition should be extended to explicitly define project management considerations as an activity, since this is very important.

The paragraph above therefore implies that this project report researches the wider and more global issues that are related to the whole life-cycle of building object-oriented component-based real-time systems.

For my employers, this programme is expected to continue way beyond the time and scope constraints intended for this project report. For this reason it is necessary to clearly define the scope of this Project Report:

* To research, manage and guide the design of the structure of the components of the various subsystems, and to direct, monitor and manage the software development process and the development team.

* To plan, formulate and control other relevant issues, including resourcing, training, quality assurance and testing.

* To record problems, analyse issues and draw conclusions on the object-oriented building block approach as applied to the development of a real-time distributed applications.

1.6 ORGANISATION OF PROJECT REPORT

Chapter 1 summarises the statement of the problem, the purpose of the study, and the organisation of the remainder of the project report.

Chapter 2 reviews the different approaches to building real-time software systems, comparing and critically analysing the issues between the fully custom-developed systems to complete product-based systems. These issues include the technical, cost, project management and risk factors. A motivation for a component-centred approach is then put forward, where re-usable components or building blocks are used to build software systems. The functional decomposition or structured analysis paradigm traditionally used to develop software systems (whether component-based or customised), is then critically reviewed and the failings are discussed.

Chapter 3 introduces the object-oriented paradigm as an alternative approach (to the more traditional functional decomposition approach) to building software systems. The new paradigm's suitability to developing re-usable components is also motivated. The relevant technologies, standards and other issues affecting the adoption and success of the object-oriented approach are reviewed. The chapter concludes with a review of software engineering and other organisational issues that had to be considered at the very beginning of the development project.

Chapter 4 firstly defines what is meant by real-time distributed systems and describes typical application areas. The generic requirements for these systems are reviewed, and in this context the design of the generic architecture is introduced. The underlying communications highway design is reviewed, and the 3-layered object-oriented component philosophy is introduced. Within the 3-layered philosophy, the lower-layer generic components are detailed for each of the software subsystems.

Chapter 5 first describes the Telecommunications Network Management (TNM) application domain. For this domain, the second-layer domain-specific components for each of the software subsystems are detailed. Chapter 5 also describes the technical details and

application engineering involved in utilising these building blocks in a real-world telecommunication network management application.

Chapter 6 is a critical analysis of the object-oriented component-centred approach. All related issues are examined, specifically the degree of re-usability and productivity, System development, project management and other organisational issues are also questioned and conclusions are drawn.

Chapter 7 concludes the study with a review of the objectives and major arguments of the study. A summary of the findings and results is presented, together with concluding statements.

Chapter 8 details areas where future research is required.

1.7 RESOURCES AND ENVIRONMENT OF THE STUDY

The material required for the research and study has been obtained from extensive literature surveys, specifically current journals, publications and conference proceedings. Material and information has also been obtained from conferences and training courses, and interaction with other companies and interest groups. Most importantly, much data and other material has been obtained from the development project itself, and from implementing and managing previous software projects.

The practical implementation and research of the study has been conducted at the Author's employers' premises. Resources utilised included a network of Hewlett-Packard 9000/800 series computers and HP9000/700 series workstations with C++, MOTIF/X, public domain software, peripherals and personal computers.

1.8 INTRODUCTION TO THE TELECOMMUNICATIONS NETWORK MANAGEMENT DOMAIN

Extensive reference is made in this report to technical real-time systems, and the Telecommunications Network Management (TNM) domain. The context of 'technical systems' and 'real-time' systems has been previously defined in this chapter.

However, to fully appreciate the subsequent material in this report, this section contains an *introduction* to the real-world domain of telecommunications networks applicable to this study, as well as the generic functions required of telecommunications network management systems.

1.8.1 Differentiating Telecommunications Network Management from Network Management

The general industry definition of 'Network Management' concerns management of entities on an organisation's corporate or operational network, and involves management of network elements such as routers, bridges, servers, hubs and other network resources. This is quite different from the requirement for telecommunications network management applications, the subject of this study. TNM involves monitoring and controlling network elements in the outside world, beyond the data communications network that inter-links the hosts of the distributed systems. Examples of these network elements include microwave, switching and telecommunications transmission equipment.

1.8.2 The Physical Model

Figure 1.1 below depicts the three relevant areas of telecommunication networks, that is, microwave, transmission and switching.

Microwave - Carries the majority of telecommunications traffic, and microwave electronic equipment has outputs that indicate the operational and error status of that equipment. These indications are monitored and stored by the remote outstations situated in the carrier rooms, and are periodically polled by nationally distributed TNM front-end systems.

Transmission - This consists of 2, 8, 34 and 140 Mbit stream multiplexer and demultiplexer equipment, which also has outputs indicating the operational and failure status of that equipment. These are also monitored by remote outstations which are periodically polled and monitored by regional TNM front-end systems. The transmission and microwave subsystems are merely bearers of traffic they are carrying.

Switching - These are the end users of the transmission and microwave systems, and outstations for monitoring this equipment are situated in the switching exchanges. These outstations periodically poll and monitor the exchange equipment, and monitor not only the faults and status, but also measures the amount and integrity of traffic being piped into the transmission multiplexer equipment, and the related traffic received at the other end from the transmission demultiplexer equipment.

1.8.3 Generic TNM Functions

From the above domain overview, it is easy to appreciate that such extensive transmission equipment requires integrated and effective management. With successful telecommunications network management, a high level of service is ensured to all users of telecommunications, from private telephone subscribers, to high-speed corporate/ banking customers.

So that the reader may better appreciate the structuring and design of the components described in later chapters of this study, the generic functions of TNM systems are summarised below:

- Monitor equipment failures on microwave, transmission and switching equipment (this will be referred to as Fault Management)
- Monitor statuses and maintain configuration on microwave, transmission and switching equipment (to be classified as Equipment Management).
- Monitor the grade (bit-error-rate) on the bearers or channels (also Equipment/Performance Management).
- Dynamically add, delete and reconfigure the telecommunications network configuration on the TNM system (Configuration Management).
- Monitor traffic (usage and utilisation) being carried by different bearers and links (this will be referred to as Traffic Management).

1.8.4 TNM Systems

A TNM system typically consists of geographically distributed computers interconnected by high-performance networks. Depending on several factors, some of the computers may serve as regional front-ends, others as Man-Machine Interfaces (MMIs), and others as central database servers. The regional front-ends typically connect to the equipment via direct or multi-dropped RS-232, X.25, Diginet etc. (Refer to Figure 6.5 in Chapter 6 for an example TNM hardware topology).

The above concludes the overview of the telecommunications network domain, including what is meant by microwave, transmission and switching, and the physical relationships between the different types of equipment. This overview has been included so that the reader can better appreciate later references to distributed real-time systems and the TNM application domain. Chapter 5 contains further details on the TNM application domain.

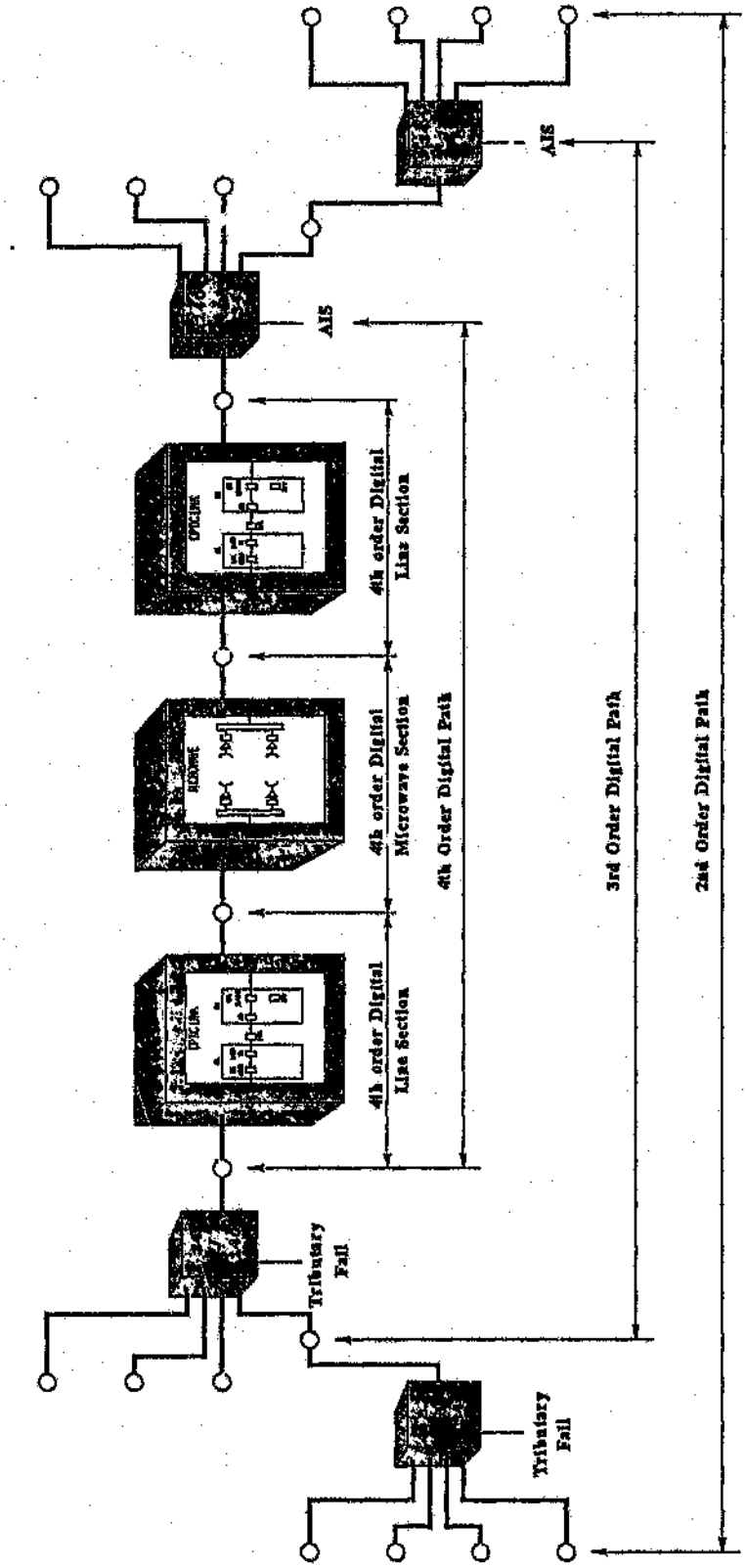


Figure 1.1: The Telecommunications Networks Physical Model

2. MOTIVATION FOR AN OBJECT-ORIENTED COMPONENT APPROACH

This chapter reviews the different solutions to implementing real-time software systems for end-user requirements. This is necessary to clarify the meanings and differences between customised, component, packaged and product-based software solutions. The issues of each of the solution approaches are compared and critically analysed.

These issues include the technical, cost, project management and risk factors. A motivation for a component-centred solution is then put forward, where re-usable components or building blocks are used to build software systems.

The functional decomposition or structured analysis paradigm traditionally used to develop software systems (whether component-based or customised), is then critically reviewed and the failings are discussed.

The arguments put forward in this chapter form the foundation for the motivation of an object-oriented component-based approach which is introduced in Chapter 3.

2.1 SOLUTIONS TO BUILDING SOFTWARE SYSTEMS

The approaches to building technical software systems, specifically real-time systems, range from the fully customised solutions to packaged solutions. These different approaches to providing solutions are briefly reviewed here.

A graph which depicts the relationship of the different solution approaches with respect to re-usability and development cost is shown in Figure 2.1 below.

The graph in Figure 2.1 (which is based on personal observation and not empirical data), essentially highlights the fact that the more customised a solution is, the higher the degree of fit but also the higher the relative cost.

2.1.1 Customised Solutions

The customised method of building software systems involves custom bottom-up software development. The development is customised around the specific requirements of the application. Minimal software is re-usable from previous applications and likewise new custom development produces little re-usable software for future applications.

The main advantage of this approach is the high degree of fit to customer requirement (provided the requirement specifications were accurate and thorough). The disadvantages include relatively high costs, long development cycles and high risk. Also a very low degree of re-usable software is produced by custom development. Further valid argument is put forward by Ahmad et al[16] and Meyer[40] in that the customised solution is highly inflexible, and thus difficult and costly and change and maintain.

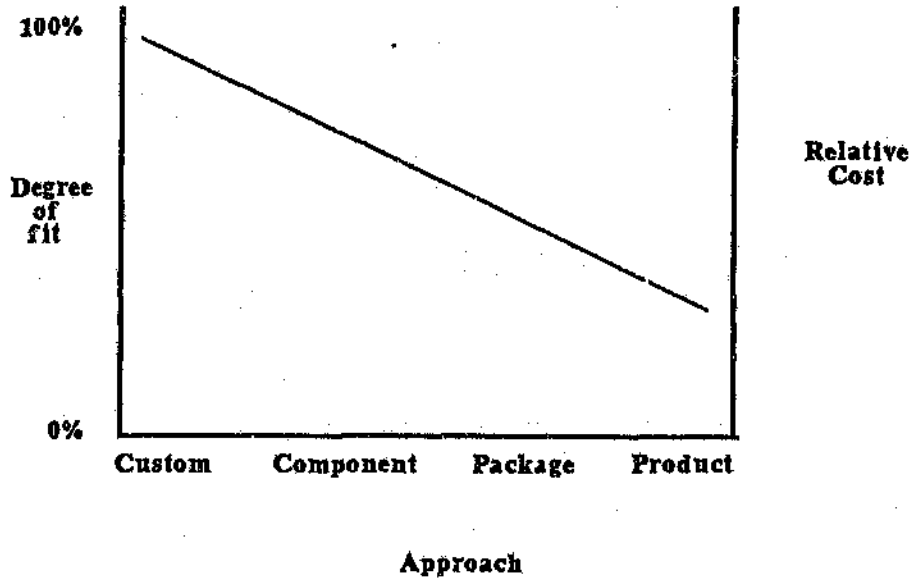


Figure 2.1: Approaches to Building Software Systems

2.1.2 Product Solutions

As seen in Figure 2.1, the product solution is the extreme opposite of the customised solution. The product solution is sold as any manufactured product and has a high level of re-use. It also has the lowest cost and risk, but in terms of disadvantages it has the lowest degree of configurability and fit.

This is not generally true of commercial products, especially with personal computer software products, but the situation is very different for product solutions for technical real-time applications.

2.1.3 Packaged Solutions

The packaged solution may be viewed as a 'flexible' or 'configurable' product solution. A solution can be provided by integrating and configuring the package or packages to provide a fit with the customer requirement. The installation and configuration of technical packaged solutions is not always customer-friendly, and generally requires a domain expert or integrator.

The packaged solution is almost a product and has a high degree of re-use for its intended application domain. It is also substantially cheaper than the customised solution. But the degree of fit is often poor. Because of its generic nature (ie. provides solutions for all requirements in its application domain), it often suffers from poor

performance and unwieldy operation and configuration. However, this approach is often a valid one, especially in cases where the package closely meets the end-user requirement.

2.1.4 Component Solutions

The component solution may be defined as functional software components each of which encapsulate particular and associated functional behaviours. This solution is positioned between the pure customised approach and the package solution. In fact, customised developments try to a greater or lesser extent to make use of components from previous applications.

Unfortunately, this re-use has not always been that effective when the software has been developed with the traditional functional decomposition/procedural programming methodologies.

Although this approach still provides a good fit to the end-user requirement, the degree of re-use is relatively low, and the cost of implementation is still relatively high.

It is in this particular area (ie. component-centred solutions), where it is strongly argued by authoritative authors like Cox[20], Meyer[46] and Booch[13] that the object-oriented paradigm makes a major contribution. It has been argued that this paradigm is far more suited to component re-use because of the very characteristics of the paradigm.

It is this very issue that is argued in this study. Later in this report it is demonstrated that the component-centred approach based on the object-oriented paradigm results in a good degree of fit coupled with the cost and risk advantages of a 'product' solution.

The different approaches to providing solutions have been outlined above, and the advantages and disadvantages of each have been reviewed. It is also evident that although it would be ideal to provide product-based solutions for technical real-time applications, the unique requirements of different applications (even in the same application domain) make this difficult.

The best compromise would be to address component-based solutions, and investigate effective methods to improve re-usability and lower the development cost.

2.2 THE NEED FOR A COMPONENT-CENTRED APPROACH

In this section the need for a component-centred approach is motivated and some of the concerns regarding marketable re-use are presented.

There has for a long time been a strong call to change the software development process from the custom cut-to-fit craft to a manufacturing enterprise. Cox[20] calls for the assembly of software solutions from robust off-the-shelf reusable subcomponents. These subcomponents are in turn supplied by multiple lower-level software producers.

Cox also talks of the 'software industrial revolution' where the key element is to create a standard parts marketplace where problem-solvers can purchase low-level pluggable

software components to assemble higher-level solutions.

However this goal is optimistic, and even Cox admits that software products are not tangible as are manufactured products. Although this is the ideal goal, there are many problems in achieving it. In this study, where this component-centred approach is researched, it was found to be a particularly difficult goal to achieve. This is mainly due to the fact that real-time technical applications possess unique and constantly changing requirements.

One of the challenges of this study was to observe how generic and re-usable the components would really be. Many articles, papers and related studies are great proponents of the object-oriented goals of high re-use, and building new applications from large stable object component repositories (eg. Gibbs[27], Arango[3]). Others such as Anderson[2] and Biggerstaff[10] have challenged the market re-usability of software components.

This state of re-usability has already been achieved in certain application areas, such as graphical user interfaces and CMIP/S communication protocols, where standard objects and class libraries exist. However, in this study it became clear that this state of re-usability would not be fully realisable - especially in technical real-time applications, where needs and requirements are complex, unique and changing.

One of the most difficult challenges in this study was to model and structure the components in a way such that a satisfactory degree of product reuse would be possible, but at the same ensuring a good fit to the customer requirement and more important ensuring adequate performance. It was important to avoid ending up with a toolbox of components that were over-generalised, difficult to fit to requirement, and cumbersome and unwieldy in terms of performance.

In addition to the goal of a component-centred approach, it was also vitally important that this approach did not compromise any of the goals and principles of sound software engineering (Ross et al[50]).

Of the different approaches to providing solutions it is therefore clear that the component-centred method is best for enhancing re-usability with the object-oriented paradigm.

In Summary

The previous sections have motivated the use of a component-centred approach to building real-time systems. This is in contrast to the use of the fully customised 'bottom-up' approach. The main benefits to be gained from this approach would be improved software re-usability and hence improved productivity. (One must bear in mind that component-based systems could be developed with either the traditional paradigms such as functional decomposition and structured analysis, or the newer object-oriented paradigm).

It has also been concluded from research literature that the object-oriented paradigm is clearly the better methodology to use for component re-use. But it is nevertheless necessary to review why the more traditional paradigms are criticised as being unsuitable.

2.3 CRITICAL ANALYSIS OF THE TRADITIONAL PARADIGM

Before an object-oriented component approach to building these systems can be motivated, it is important to critically analyse the traditional approach and review the typical problems and shortfalls. The more important issues are discussed here.

2.3.1 The Traditional Paradigm

The traditional paradigms utilise the principles of structured analysis and design, functional decomposition, and the 'waterfall' software life-cycle with its project phases. This includes the definition of the User Requirement Specification, Functional Requirement Specification, the design phase, module design, pseudocoding, coding, testing, integration, commissioning and acceptance. This approach to software engineering, which is widely practised, is well documented by noted authors including De Marco[21] , Yourdon[57], Gane and Sarson[25], and Ward and Mellor[53] who have extended the basic methodology with real-time extensions.

2.3.2 Modelling of Functions

Generally structured analysis and design of non-commercial systems places emphasis on the modelling of functions, and little emphasis on data modelling. As Yourdon[58] observes, more recent variants of structured analysis do focus more on data with the entity-relationship diagrams, but many project teams ignore them altogether when modelling user requirements, focusing instead on the ubiquitous data-flow diagram.

2.3.3 Not Suited to Graphical User Interfaces

This problem refers to the fact that structured analysis provides little or no guidance in developing the user interface of a system. As Yourdon[56] commented, structured methodologies were developed in the 1970s before the advent of the GUI (Graphical User Interface). These days often a major portion of the software is associated with the GUI.

2.3.4 Changing Requirements

The traditional top-down approach takes no account of evolutionary changes. This is observed by Lehman[42] who classifies programs into S, P and E-programs, of which P and E programs indicate software evolution being an intrinsic, feedback-driven property of software. The functional decomposition approach does encourage a degree of information hiding and generalisation through the use of functions and subroutines. But because functions are still modelled around globally accessible data storage areas, any changing user requirement that affects data structures has a domino effect on the rest of the subsystem. So this approach results in systems with a low degree of robustness.

Experience has shown that changing requirements are a fact of life in software engineering. How often has it not happened that a new requirement forces the designers to back-track to earlier stages of the life-cycle. Often a change perceived as minor has extensive implications because functions are modelled around common data structures.

2.3.5 Software Engineers lack Domain Knowledge

Closely related to the above issue, and aggravating the situation, is the lack of domain knowledge by software engineers. Software engineers are computer literate and can implement the requirements as perceived. However they are not necessarily domain knowledgeable, and may not understand and correctly interpret the user's real requirements.

There has been a strong call, particularly with the object-oriented paradigm, for a domain specific approach in the software community[26,46,34].

Gibbs et al[27] have already proposed the ideal scenario where applications would be based on generic software components accumulated by a software community familiar with the application domain. Pliskin et al[48] have taken this issue further by proving that rapid prototyping by end-users enables more effective utilisation of domain knowledge. With the traditional paradigms, it is evident that it is more difficult to separate the functions of implementing generic software and implementing domain specific software. Hence it is necessary for developers to have the necessary domain knowledge as well. As will be demonstrated later in this study, the object-oriented paradigm is well suited to separating these roles. On this basis, not all developers of projects need to have the domain knowledge.

The importance of domain knowledge should never be under estimated. At the start of this study, it was required to develop additional domain engineers to assist with development of the domain-specific components. It proved to be particularly difficult to gain knowledge of the complex telecommunications network management domain. In some cases it took newcomers several months just to learn the new domain vocabulary, and took several more months to understand the domain function models.

2.3.6 Inflexibility of Top-down Design

Another criticism levelled at the top-down approach is its inflexibility. As Meyer[46] argues, top-down design may be effective for developing individual algorithms and routines, but goes against re-usability when applied at the systems level. The reason is that top-down promotes one-of-a-kind development and relies on a single, frozen top-level function - a case that is rare in practise.

2.3.7 Not Suited to Re-use

A major problem of the structured approach is its unsuitability for re-use. As correctly observed by Edward Yourdon[50], the programming and documentation notation of structured analysis provides little mechanism for emphasising re-usable components. The reasons are mostly due to the enormous emphasis placed on the modelling of functions, with less emphasis on the data modelling. The result of this is difficulty in abstracting function models with clear interfaces.

Another interesting comment made by Meyer[46] is that the top-down design method goes against the key factor of software re-use, because it promotes one-of-a-kind developments, rather than general purpose combinable software elements.

2.3.8 Decreasing Hardware Costs highlights Software Cost

Another factor is the ever decreasing cost of computer hardware which further highlights the software crisis. In some past projects in our organisation, hardware has accounted for as little as 25% of the total contract value. It is becoming more apparent that software is too costly, and of insufficient quality, and its development is near impossible to manage.

2.3.9 Estimation of Time and Resources Required

Another problem characteristic of the 'waterfall' software life-cycle of the traditional approach is accurate estimation of cost, time and other resources required for software development and maintenance. Techniques available for estimation of resources range from very theoretical empirical methods to crude estimation based on past projects. In general, as observed by Lehman[40], techniques estimating project requirements based on objective measurement of such attributes as application complexity and size have proved unsatisfactory.

It is of interest to note that this study proved that the object-oriented approach proved no better in this regard. This issue is discussed in detail in a later chapter.

2.3.10 Effort of Maintenance

Basili[6] neatly defines the following maintenance activities: correcting faults in the system, adapting the system to a changing operating environment, and adapting the system to changes in the original requirements.

The maintenance problem of traditional data-structured software is well known and documented[7,35,40,57]. The main criticism is that any change to data structures affects all processes and functions associated with the data structures. This generally results in the maintenance activity costing more effort than the original development activity.

What makes the maintenance issue even more important is the sheer size of this activity. A statistic quoted by Lehman[40] indicated that in the U.S.A, 70 percent of programming activity was spent on program maintenance, and only 30 percent was spent on program development.

Another study by Wilma Osborne of the National Bureau of Standards in the U.S.A. substantiates the above statistic by suggesting that 60 to 80 percent of the total cost of software is due to maintenance[47]. Since one way to re-use a program is to enhance it, maintenance is a special case of software re-use.

2.3.11 Other Project Management Issues

In our organisation's experience, the relatively long project life-cycles of the traditional approach creates other problems such as team demotivation and staff turnover. Also, the sometimes long periods of system commissioning at remote sites presents additional problems.

The long project life-cycle also has financial implications. When developing for the end-user, it is difficult to motivate progress payments for certain milestones. Some customers do not regard the specification, analysis and design documents as 'products', and progress payments can often only be negotiated for written software that can be demonstrated. This may have implications on the project financials and cash-flow when running the project over several calendar years, as demonstrable implementation only proceeds only well into the project life-cycle.

2.4 CHAPTER SUMMARY

In this chapter, two issues were discussed. The first was a review of the different methods of providing software solutions, and the second was an analysis of the criticisms levelled at the traditional procedural approach to developing software systems.

Of the different solution approaches, it was found that the packaged and product-based solutions were not ideally suited to the unique requirements of distributed real-time technical systems. On the other hand it was noted that the customised approach is expensive, risky and has long implementation times. However, it was motivated that the component approach is where the best contribution can be made to improving the software engineering process. This is primarily due to the object-oriented methodology's suitability to building re-usable components.

In the second part of the chapter some of the criticisms levelled at the traditional functional decomposition paradigm and the associated 'waterfall' life-cycle were reviewed.

Based on the arguments reviewed in this chapter, it was believed that there is a strong motivation for an object-oriented component-based approach to building software systems, and later in the study the results, analysis and criticisms of this approach are recorded.

3. THE OBJECT-ORIENTED PARADIGM AND RELATED ISSUES

In the previous chapter, the need for an object-oriented component-centred approach to building software systems was motivated. The objective of this chapter is to firstly introduce the object-oriented paradigm and then to review and debate some of the related considerations. This is necessary before leaping directly into the design of the core components.

Firstly, this chapter introduces the object-oriented paradigm as an alternative to the more traditional functional decomposition approach to building software systems. The introduction reviews the basic principles and characteristics of the object-oriented paradigm.

The relevant hardware and software technologies, standards and other issues affecting the adoption and success of the object-oriented approach are then detailed.

Because the object-oriented life-cycle is quite different from the traditional 'waterfall' life-cycle, the object-oriented life-cycle found appropriate to this project is described. The model used for the development team roles is also described.

Finally the chapter concludes with a review of software engineering and other 'soft' organisational issues that had to be considered at the start of the development project.

3.1 THE NEW OBJECT-ORIENTED METHODOLOGY

The origins of object-orientation can be traced as far back as the early 1960s with languages such as ALGOL 60 and Lisp. Since then there has been a steady evolution of object-based languages such as Pascal, Ada and C and true object-oriented languages such as Simula, Smalltalk, C++, CLOS and LOOPS (Booch[13]). These languages possessed many of the characteristics of object-orientation as we know it today. However it appears that these origins were more language based, and it was only in the mid 1980s that object-oriented was first formalised as an actual methodology for constructing software systems, and included OOA (object-oriented analysis) and OOD (object-oriented design) as well as OOP (object-oriented programming).

From the mid 1980s it has progressed from being an impressive buzzword in the software marketplace to a more mature and developed methodology that is steadily gaining acceptance and being more widely applied.

There exist many good books and article references on this methodology, and it is beyond the scope of this study to detail the methodology itself. However the basic philosophy of the methodology is outlined here, and the reader is encouraged to refer to the bibliography of this study for several good references on the object-oriented paradigm.

The object-oriented methodology is said to be a new method of viewing software, where the emphasis is shifted to the objects we build rather than the processes we use to build

them. Object-orientation means abandoning the process-centric view of the software universe where the programmer-computer interaction is paramount in favour of a product-centred paradigm driven by the producer-consumer relationship.

It allows better factoring of functionality and related data into components that are minimally coupled. The data is encapsulated with its associated processing and clear interfaces are defined. This means application programmers do not have to be concerned with the processes within objects - how they work and implementation detail is not required.

The object-oriented paradigm includes object-oriented environments, databases and architectures. It also includes the analysis, design and programming methods. There exist many references and studies on the different OOA and OOD specification and notation methods. One of the references highly favoured by the majority of the development team of this project was that of Grady Booch[13], titled Object-oriented Design with Applications.

In essence, the object-orientation paradigm has been narrowly defined by Wegner[54] to contain 4 main properties:

(1) Objects have operations that define their behaviour and variables that define the state of the object between operation calls.
(2) Classes describe the common behaviour of collections of objects.
(3) Classes may be specialised by defining a class that adheres to all the behaviours of the original class. Additional behaviours and/or state variables can be defined for the new class. (this is generally referred to as inheritance, where the original class is called the superclass and the new class is called the subclass).
(4) Objects are first-class citizens and obey the semantics of other types in the language. (This is generally called aggregation, as it allows objects to be composed of other objects).

The section above on the object-oriented methodology has briefly described the origins of the object-oriented paradigm and the basic characteristics. For further details on the methodology itself and the supporting environments, refer to the bibliography.

3.2 NEW SOFTWARE TECHNOLOGIES AND STANDARDS

Although the subject on open systems, and formal and de facto standards for Common Application Environments (CAE) is very extensive, it is necessary in briefly review the current status of the industry in terms of standards and common operating environments. This is because these standards very much affect the degree to which the object-oriented components can be re-used. More importantly, the status of the standards industry directly affects the use of the CMIP-conformant software highway, which is one of the research points in this study.

3.2.1 The Move to Open Systems

While open systems mean different things to different people in different software industries, the ongoing standards battles has significant impact on the evolution of an effective open environment. Hampel[31] and Held[33] believe that the scope of the Open Systems drive includes:

* consistent user interfaces
* database systems and consistent data access
* operating systems
* local networks
* backbone networks
* inter-company networks

The primary objectives of the Open Systems drive is to guarantee portability,compatibility,inter-operability and scalability (Hayes[32], Hampel[31]). Each one of these objectives determines the degree to which object-oriented components could be re-used.

The evolution of the de facto standards and formal standards for a Common Application Environment (CAE) is quite interesting. UNIX International (UI) and the Open Software Foundation (OSF) appear to be driving the industry towards different standards, although the two bodies have several features in common (Hampel[31]).

These include:

* Transmission Control Protocol/Internet Protocol (TCP/IP) for communications
* Network File System (NFS)
* Compliance to POSIX
* Commitment to the OSI standard

However these are bodies that drive the de facto standards, and they in turn are influenced by several industry pressure groups such as the Petrotechnical Open Systems Corporation (POSC), the Int. Public Sector Information Technical Group (IPSIT) and the User Alliance for Open Systems (UAOS).

At the other end we have the emergence and evolution of the formal standards such as ISO, IEEE, POSIX and ANSI. In between the de facto and formal standards, the X/Open organisation has emerged as a bridge between the UI and OSF and the formal standards. As an independent body, it takes on the role of combining formal and de facto standards in an attempt to create a Common Applications Environment (CAE), providing portability and inter-operability.

Figure 3.1 below graphically shows the current relationship of standards bodies (Hampel[31]).

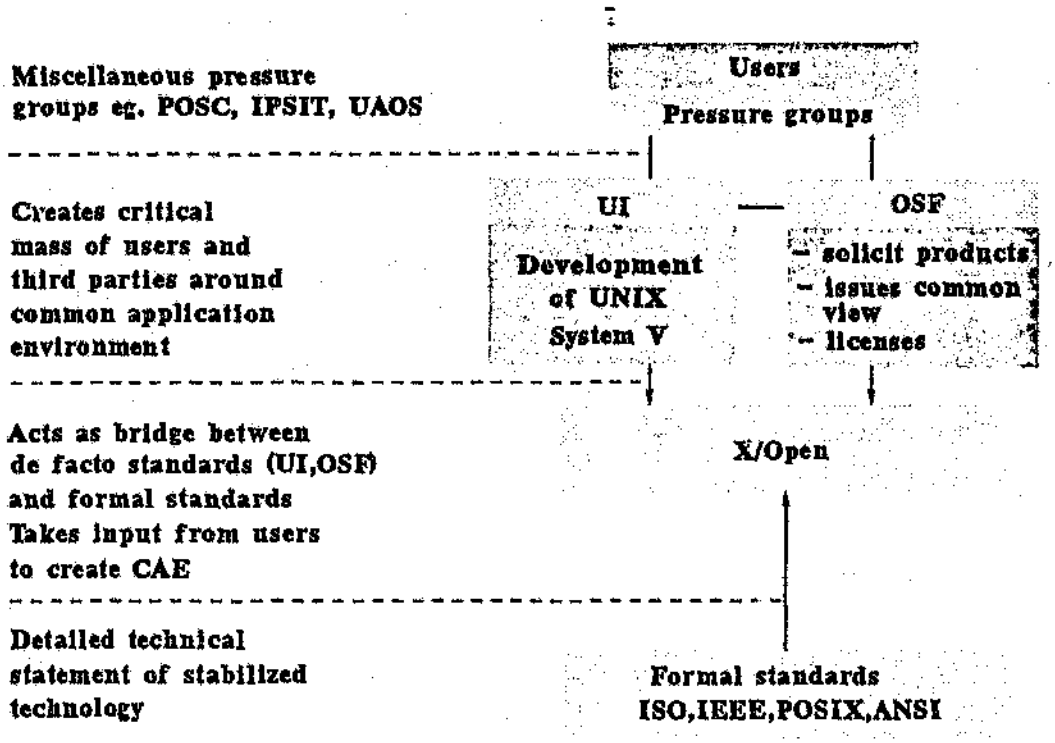


Figure 3.1: Relationship of Standards Bodies

3.2.2 Implications for the Object-oriented Approach

The objectives of Open Systems and the associated standards are seen as a benefit both to the users through greater choice of hardware platform and to software developers through availability of a volume marketplace on an international basis.

In theory the objectives of standard operating environments and inter-networking sound good and make sense. However in reality there are still many headaches for the systems developer and integrator because of the different versions and flavours of products based on the so-called standards.

Apart from technical acceptance of and compliance to standards, another problem is

end-user industry acceptance of the standards. One of the major decisions that had to be made for this project was whether to adopt the well-entrenched and proven TCP/IP protocol stack for inter-network communications or adopt the much newer OSI communications stack.

3.2.3 In Summary

It has been necessary to review the software technologies and standards applicable to this project, since they have an effect on our object-oriented component approach strategy. Since this project is not based on a single computer, but on a real-time network of heterogeneous computers, it was important to evaluate which operating environments and standards we would adopt, and which should be catered for in the future. With the right choices it was believed that the applicability and marketability of re-usable components would be wider.

Although the goals of Open Systems are common, there still exist many divergent views on standards, as observed by Hampel[31]. Although further formalisation of the standards is taking place and the formal and de facto standards are converging, the situation presents many operational and strategic problems for a software development organisation like ourselves.

As reviewed in this chapter and in Appendix A, the status of the standards industry is complex and extensive, and the material reviewed here is only the tip of the iceberg. An extensive study and evaluation of the standards industry is not in context of this study, and the reader is encouraged to refer to the relevant references listed in the bibliography. The brief review of the standards industry given in this chapter, and in Appendix A, has been sourced from the references in the bibliography and several other academic and industry-related publications.

At the end of the day, it is important to cater to the end-user demands. Currently many end-users are very much entrenched in de facto standards, although they claim to be committed to the direction of formal standards. The implication of this for our organisation is to structure our object-oriented component approach strategy and design so as to be able to accommodate and migrate to the formal or whatever standards the end-user industry adopts in the future. This strategy is evident in the philosophy and design of the object-oriented component approach as detailed in the remainder of this study.

3.3 NEW HARDWARE TECHNOLOGIES

In this chapter, we need to review the evolution of hardware technologies, since this has an effect on standardisation, portability, software complexity and the object-oriented methodology. To have all these good things extra processing power, memory and disc storage is necessary. Fortunately the evolution of hardware technology has been remarkable.

An example of this is the Hewlett Packard HP9000 range of UNIX minicomputers (shown in Table 3.1 below), whose models have shown an approximate doubling of performance and memory capacity every 14-18 months for the same relative price.

H/W Platform	Year	MIPS	Max. Memory
HP9000/825	87	8	48Mb
HP9000/832	89	16	64Mb
HP9000/827	91	38	128Mb
HP9000/720	91	53	128Mb
HP9000/730	92	75	256Mb
HP9000/735	93	120	256Mb

Table 3.1: Example of Hardware Technology Evolution

Why exactly do we require all this power for the object-oriented approach? Firstly the building block approach with object-orientation implies a high level of component encapsulation with standardised software interfaces, application programmer interfaces and a degree of redundancy in data and functionality.

Secondly, in real-time applications a great proportion of managed objects need to be memory resident for performance reasons, and use of a generic interface to the real-time software highway requires very complex multi-layered communications software. These issues are further reviewed and analysed later in this report.

3.4 THE OBJECT-ORIENTED LIFE-CYCLE

From research of relevant literature[35,29,38,40] and our own pilot prototyping, it became obvious that the life cycle of the object-oriented approach to building components would be quite different from the traditional 'waterfall' life cycle. In this section some of the problems of the traditional waterfall life-cycle are reviewed, and the model we used for the object-oriented component-based development is described.

3.4.1 The Classical Waterfall Life-cycle

The Waterfall model (which is generally attributed to Royce[51]) used in the classical procedural approach entails a unidirectional flow of development from initial requirements to final product. This is shown in Figure 3.2 below.

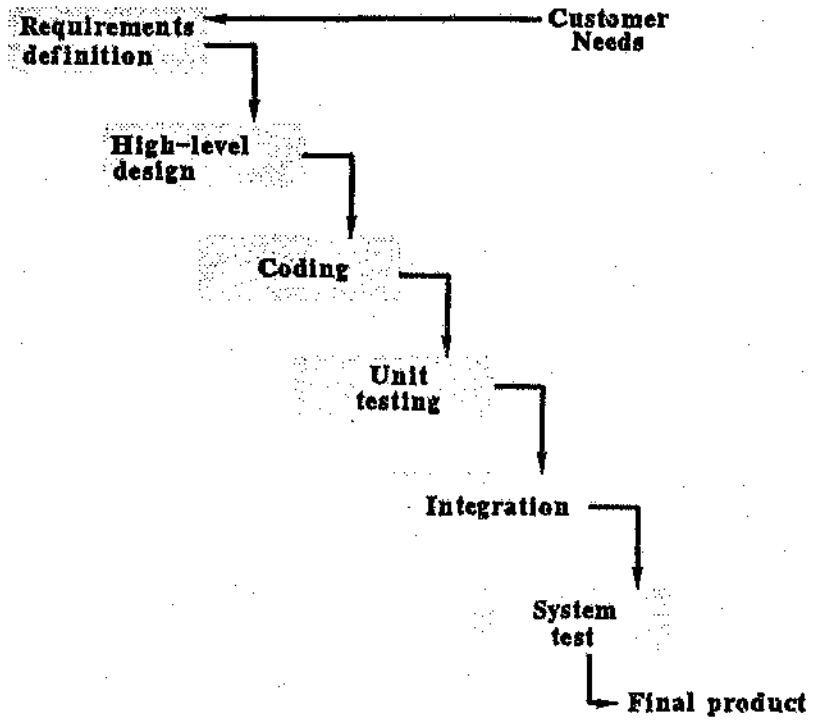


Figure 3.2: The Waterfall Lifecycle Model

There are various other refinements and adaptations to this life-cycle model, such as the seven-phased model of Boehm[11], the U.S. Department of Defence Standard 2168, and the Yourdon structured life-cycle model[57]. The problem with these approaches is that it is very costly, if not impossible to go back to a previous development stage. Often inconsistencies are only discovered in the system testing stage, and rectification at this stage is difficult and costly.

3.4.2 The Object-oriented Life-cycle

With the object-oriented approach, it has been shown that it is feasible to actually prototype and test the contemplated system before any commitment to a given architecture. Studies done by Jrad et al[36], Goldberg[29], Henderson-Sellers et al[35] and others have shown the life-cycle model for object-oriented development consists of closely related iterative steps of implementation, design and testing.

In this study, I observed a life-cycle model which is a combination of the models proposed by Jrad[36], Goldberg[29] and Henderson-Sellers[35]. This model is shown in Figure 3.3 below.

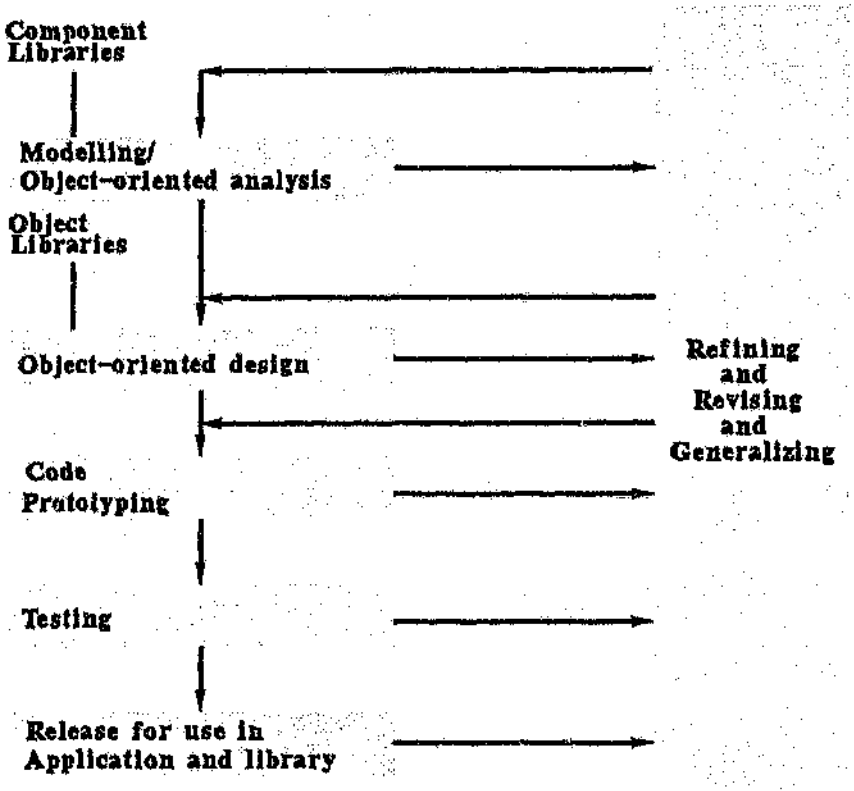


Figure 3.3: The Object-oriented Life-cycle Model

Figure 3.3 clearly shows the tight coupling between the different stages and the iterative process to developing object-oriented systems. It also shows the bi-directional flow of development. This means that when problems are found, they are always fixed in the stage where they ought to be fixed.

This section has provided a review of the object-oriented life-cycle, and examined the fundamental differences between this life-cycle and the traditional waterfall life-cycle. Discussion of the results and other issues relating to the object-oriented life-cycle are contained in a later chapter of this study.

3.5 THE DEVELOPMENT TEAM ROLES

Once we had defined, observed and further refined the iterative-prototyping life-cycle model, the roles in the project team had to be defined and clarified. Several authors, including Arango[3] and Pliskin et al[48] refer to the separation of the software development roles of toolbuilders and application engineering. In this study it was necessary to develop their models further: defining distinct roles of toolbuilder, application engineering and application consulting.

One of the more notable problems in the traditional structured approach is stated in an interesting paper by Pliskin et al[48]. They observe that *'end-users, not having any knowledge of structured techniques, were struggling to comprehend the various software engineering tools, products and methodologies. Software engineers, lacking domain knowledge in the area of the application, were struggling to seek domain knowledge from users and to comprehend the interactions between users and the system sought'.*

The use of object-oriented building blocks presented opportunities to address this problem. Right at the start I sought to separate the functions of generic component builders, domain builders, application engineers and application consultants. The main motivation for this was to optimise re-usability, particularly in this study where a specific 3-layered component structure was employed (see Chapter 4).

However, in practice it was observed that these roles did not function as mutually exclusive as one would have hoped. The application engineer was not entirely divorced from the domain and component builder functions, and much interaction and co-operation was necessary. (These issues are discussed in detail in the analysis of the approach in Chapter 6).

The development team model I have proposed is an extension and further formalisation of the models presented by Arango[3] and Pliskin[48]. The functions of generic component builders, domain builders, application engineers and application consultants are separated. A diagrammatic depiction of these roles and their interactions is shown in Figure 3.4 below, and further description of the roles follows.

3.5.1 The Component Builder

The component builders were primarily involved in building up the toolkit of *generic* objects and their associated functionality. Objects were deemed generic if it was expected to be used in other technical real-time application domains. The component builders needed to have good knowledge of the overall generic building block philosophy, deep knowledge of the object-oriented paradigm and knowledge on implementation specifics such as language.

The component builders added further objects and object-based components to the generic toolboxes, and also maintained the existing libraries of objects and object subsystems.

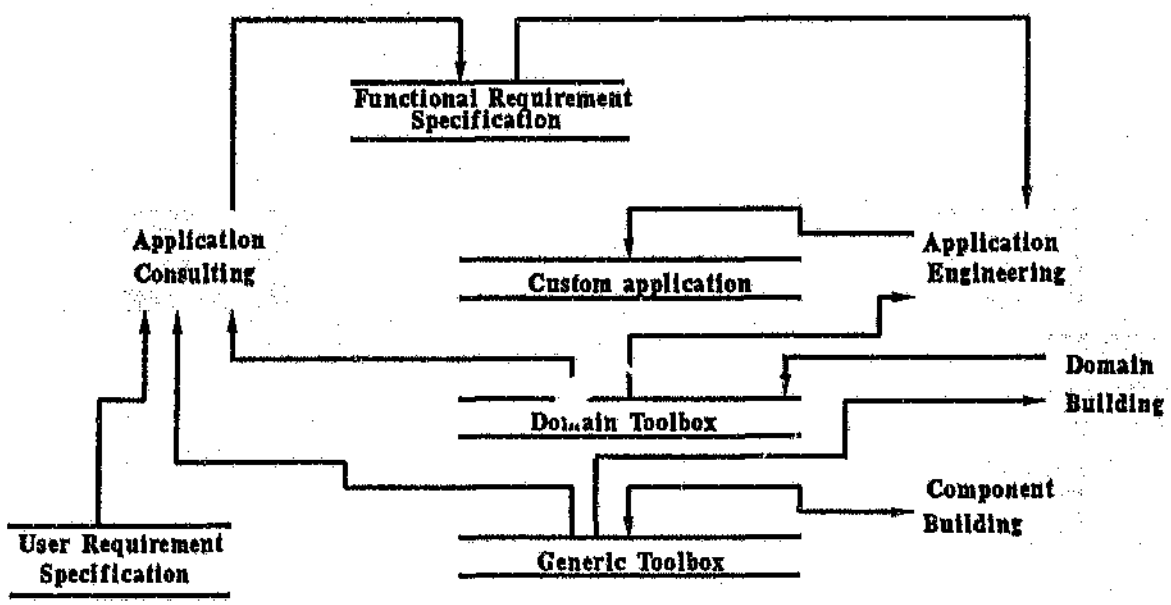


Figure 3.4: Development Team Roles

3.5.2 The Domain Builder

The domain builder maintained and added objects and object subsystems to the *domain* toolboxes. The domain toolbox contained components expected to be generic to other applications in the same domain. In this study, the domain was selected to be telecommunications network management. These components were extensions and specialisations of the generic components, with a 'flavour' for telecommunications network management.

3.5.3 The Application Consultant

The application consultant was the person or role that needed extensive application knowledge. In this project, we had two application consultants who had wide knowledge of telecommunications applications, their users and their domain specific problems and concerns.

Because of their application and domain knowledge, the application consultants were quite capable of understanding and documenting the application requirements, that is, interpreting the user requirements into a functional requirement specification. Apart from

the domain knowledge, it was also observed that the application consultant had to have a reasonable technical appreciation of the components and the features and constraints of the architecture and components.

3.5.4 The Application Engineer

The application engineer was the person who made use of the Functional Requirement Specification, the *generic* components and the *domain* components. This person then developed the application layer of the software using (re-using) the object-oriented components from the toolboxes.

The application engineers had a thorough knowledge of object-oriented design and modelling, as well as the implementation language. However, in keeping with this role, this person had little knowledge of details within the component objects - only knowledge of the services provided by the object components, and the interfaces to them.

The above 4-role model for the object-oriented analysis-design-implementation life cycle worked reasonably well. Although the roles did have areas of overlap at times, I did find that focusing on this 4-role model strengthened the whole philosophy and attitude to re-using the components. These issues are discussed in detail in Chapter 6 of this report.

3.6 ASSESSING THE OBJECT-ORIENTED METHODOLOGY

3.6.1 Achieving the Goals of Software Engineering

Before adopting the object-oriented approach, it was necessary to assess whether this was the right approach for building technical real-time systems and whether it was in line with generally accepted software engineering principles.

According to Ross et al[50], the four fundamental goals of software engineering are modifiability, efficiency, reliability and understandability. The seven principles that affect the process of attaining these goals are modularity, abstraction, hiding, localisation, uniformity, completeness and confirmability.

These goals and principles are very much accepted in the industry with regard to the traditional functional decomposition methodology. However, would object-orientation achieve the same goals and could the same principles be practised? I believed they could. The very essence of object-orientation is claimed to be maintainability through modularity, abstraction and information hiding.

The principles of uniformity, completeness and confirmability were also concerns at the start of the project. Testability and confirmability are stated as advantages of object-orientation - but in implementation we initially found it difficult to test objects and object subsystems. However this was improved with trace and view and other utilities specially written for testing and debugging. Further details and conclusions on this appear in Chapter 6 of this study.

Would the goals of software engineering be achieved? There were several references to the improved modifiability and reliability that have been observed, but I had reservations about the goals of efficiency and understandability. What was particularly alarming was

the observation by Jrad et al[36] that execution time of object-oriented software in a real-time system proved to be three times that of procedural software.

As far as understandability was concerned, the project team's initial introduction to object-orientation proved that understandability was an issue. It required a great deal of effort to get 'traditional' developers and designers to truly understand the object-oriented paradigm.

3.6.2 Prototyping and Incremental Development

Systems development using the waterfall development methodology is usually a slow process, in which the turnaround time from concept to commissioned system could take up to several years. To take on a brand new methodology like object-orientation in this fashion appeared to be very risky. It was felt that incremental development and rapid prototyping was the correct approach to adopt (Jacobson[30], Goldberg[29], Jrad et al[36]), at least for the initial development of the re-usable building blocks.

This advice was taken seriously. Eight months before the main development project started, a C++ compiler was loaded onto a UNIX workstation, and two software developers were set the task of developing a very simple prototype graphical user interface using object-orientation. This was our 'pilot' system, to gain a better understanding of object-orientation principles, how to apply it, and better understand the constraints.

Based on the success of this experience, it was decided to approach the main development of the object-oriented components in the same manner. The components would be developed incrementally step-by-step, beginning with the core of the system and a few of the more important system functions. Components were then rapid-prototyped and refined, and then in the same way the next component was developed. In this way the system was incrementally enlarged.

3.7 PROJECT MANAGEMENT AND CONTROL

It was not the intention of this report to explore all the project management aspects of the object-oriented approach. This section briefly reviews the project management methods, and tools used for management and control of the project.

In terms of methods, the same methods as used for the traditional approach to building software systems were used. Formal methods included weekly progress meetings, scheduled technical discussion sessions, and quality assurance and auditing work-groups. Informal methods included ad-hoc technical meetings and integration sessions.

The tools used were also similar to those used for the traditional approach. Two primary tools were used:

(a) The internally developed Management Information System (MIS) which was used for recording of progress as per budgets, and full financial control of the project.

(b) Symantec's ONTARGET project planning package was used for tracking of progress on individual subsystems and components, and for task and resource

scheduling and planning.

3.7.1 The Management Information System

The MIS is a SCO UNIX-based 80386 PC running the ORACLE relational database management system. It reports the following:

* Recording of hours logged (on a weekly basis) by the development team, and monitors progress as a percentage of logged hours to budgeted hours.

* All income accrued and costs incurred by the project, and net profit calculations. Income would typically be income received as a result of sales orders, or received as 'internal' incomes from other projects. Costs incurred include manpower costs and other organisational overhead costs.

3.7.2 The Project Planning Package

Symantec's ONTARGET package was used and proved to be successful in combination with the MIS. It is a PC-Windows based package and provided detailed itemisation of Work Breakdown Structure (WBS) tasks, allowing tracking of progress, task dependencies, resource scheduling, and an extensive user interface via PERT and GANTT charts.

The MIS provided tracking of manpower effort and other expenses against budgets, the Project Planner provided tracking of progress and task and resource scheduling. Examples of reports provided by these tools are contained in Appendix B of this report.

In terms of the contribution of these tools to the success of the object-oriented approach, the MIS and Project Planning Package worked reasonable well, but did have shortfalls. Problems were experienced with accurate tracking of progress on the object-oriented components being developed. This proved to be due to the unique characteristics of the object-oriented paradigm. This was no direct fault of the tools used, but it would be interesting to note whether tools ideally suited for object-oriented project management become available in time to come. This issue is reviewed in greater detail in Chapter 6 of this report.

3.8 COMPONENT MODELLING AND DESIGN METHODOLOGY

This section reviews the object-oriented analysis and design methodologies used in this project. The documentation techniques are also reviewed and a modified hybrid of some of the better known techniques is demonstrated.

Some of the earlier object-oriented development methodologies were not generic, having being structured specifically for the Ada environment. In recent years however, several non-Ada oriented methodologies have appeared. Meyer[45] focuses on class development using formal specification of abstract data types (ADTs) to define behaviour. He also provides examples on how to transform a functional design to an object-oriented design.

Coad and Jourdon[18] describe an OOA method using data modelling techniques which define system architecture in terms of assembly and classification structures. Services are

specified for objects by techniques such as entity-life-history and state-event-response charts.

Wirfs-Brock and Wilkerson[56] use a responsibility-driven approach which concentrates on the responsibilities of an object and the information it shares with other objects. The objective of this approach is to increase encapsulation of information within an object.

3.8.1 The Object-oriented Development Process

After evaluation of the different development methodologies, it was decided to use the Booch[13] method as a basis. Booch suggests starting with a natural language description of the desired system and using the nouns as a starting point for the classes of objects to be designed. Each verb is an operation. The operation is implemented by a class when the class is the direct object, or the operation is one used by the class when the class is the subject. This resulting list of classes and operations can be used as a starting point in the design process.

We found it essential to extend the Booch approach with the approach described by Johnson and Foote[37], which not only defines classes for *objects* in the problem domain, but where required, define classes for *operations* in the problem domain. Further experience and the degree of abstraction dictates whether an operation should be a method in a class or a separate class.

It is well accepted that the OOD process is neither top-down or bottom-up, but is an iterative and incremental development of a system. This is due to the refinement of different yet consistent logical and physical views of the system as a whole. This is the essence of the Booch approach[13] which can be summarised in 4 steps:

(1) Identifying the classes and objects at a given level of abstraction
(2) Identifying the semantics of these classes and objects
(3) Identifying the relationships between these classes and objects
(4) Implementing the classes and objects

In this project, the Booch Class Diagram and Object Diagram were two notation methods which were extensively used for depicting the static semantics of the design. It was observed that in each of the above steps, the class and object diagram products of the previous step were further refined and modelled.

It was found that the Booch state transition diagrams and timing diagrams were particularly useful for illustrating the dynamic semantics of the real-time components. Timing diagrams proved particularly useful in illustrating the timing and order of events among associated sets of managed objects. Module diagrams helped illustrate physical packaging of classes and objects into program modules.

(In this study, the term 'modelling' is used. I have defined modelling to be the next step after the object-oriented design and development process. Modelling is subtly different from OOA/OOD, in that modelling determines which objects initiate activities and the sequence of activities, the lifecycle of objects and the state at different parts of the cycle).

3.8.2 The Object-oriented Development Notation

Class and Object diagrams were extensively used in the project for the object-oriented analysis and design process. It was necessary to use both these notation methods to document the logical designs, since each illustrated entirely different design decisions. Class diagrams were used to document the key abstractions of the system, and object diagrams documented the mechanisms manipulating these abstractions. In a sense, object diagrams captured the dynamic semantics of operations, and represented the interactions that may occur among a collection of objects, no matter what specifically named instances participate in the mechanism. Figures 3.5 to 3.8 illustrate the various object-oriented design/notation methods that were evaluated.

A subset of Booch's notation standards were defined for this project, and certain extensions were made. Examples of the extensions include the notation to depict the software bus CMIP-conformant messages between objects, and notations to depict UNIX inter-process messages and QCOMM-specific messages. These notation extensions are depicted in Figure 3.11. The modelling aspects of the subsystems were notated using extended object diagrams, state transition diagrams and timing diagrams.

The following figures illustrate examples of the various object-oriented design and notation methodologies used on the project.

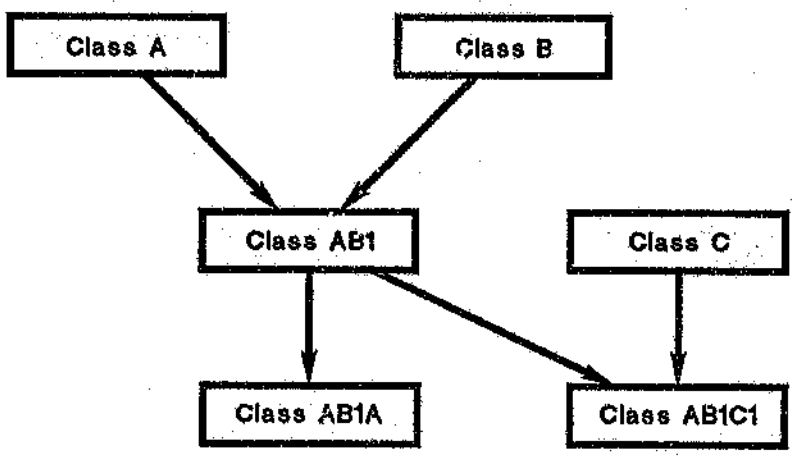


Figure 3.5: The Responsibility Driven Design Hierarchy Graph (as proposed by Wirfs-Brock[49])

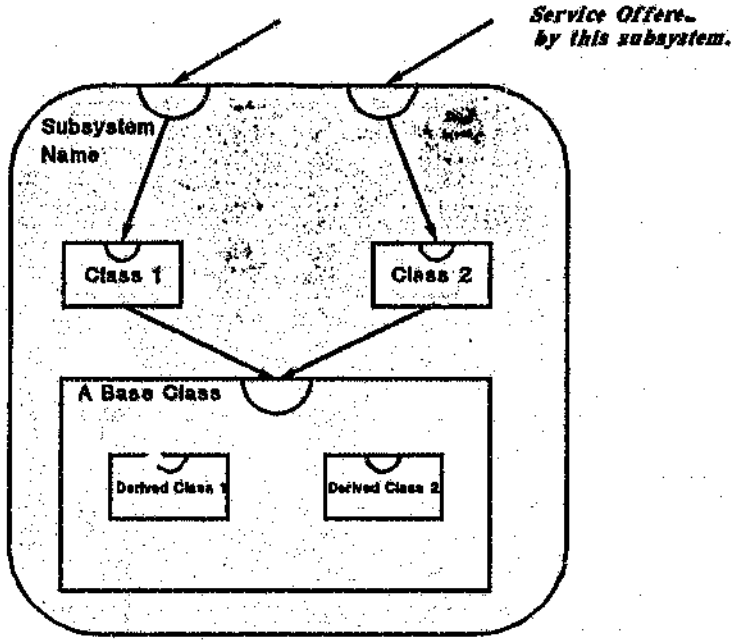


Figure 3.6: The Responsibility Driven Collaborations Graph (as proposed by Wirfs-Brock[49])

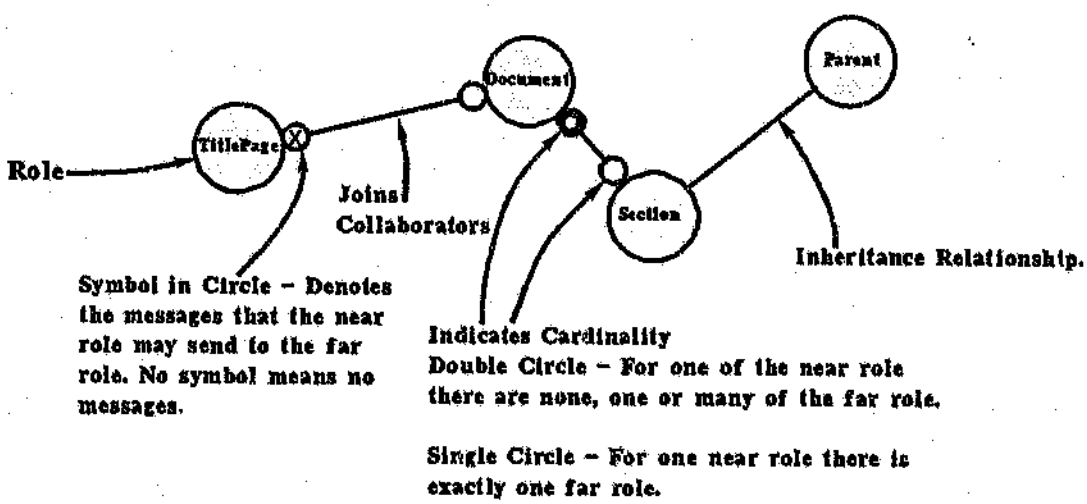


Figure 3.7: The Role Model Diagram (as proposed by Wirfs-Brock and Johnson[62])

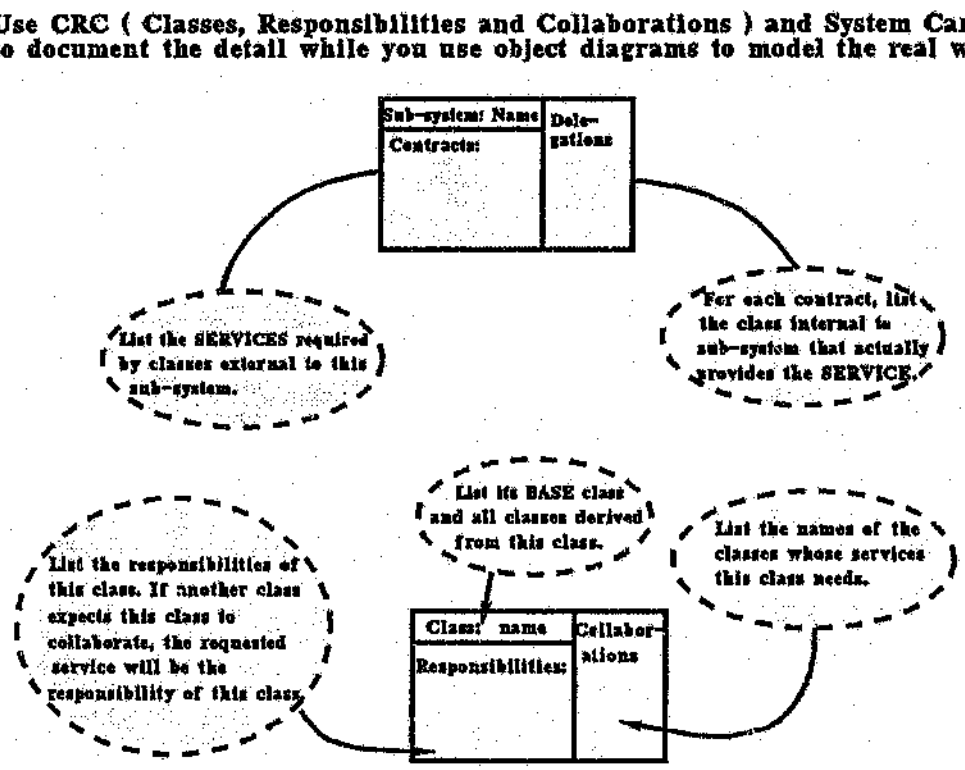


Figure 3.8: The CRC Cards (as proposed by Beck and Cunningham[88,57]).

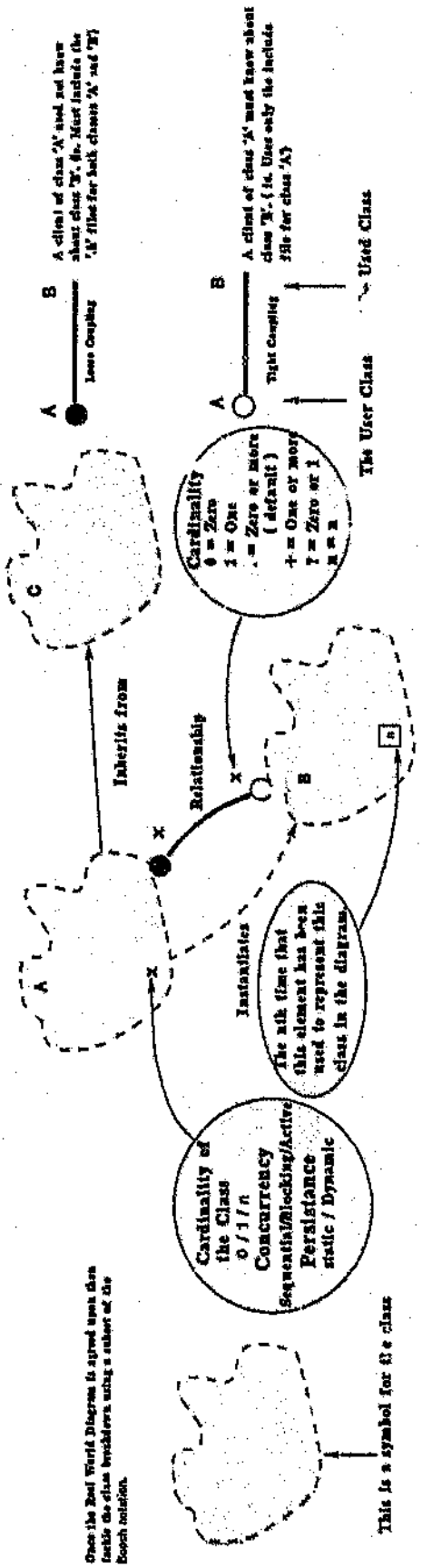


Figure 3.9: The Class Diagram (as proposed by Booch[45])

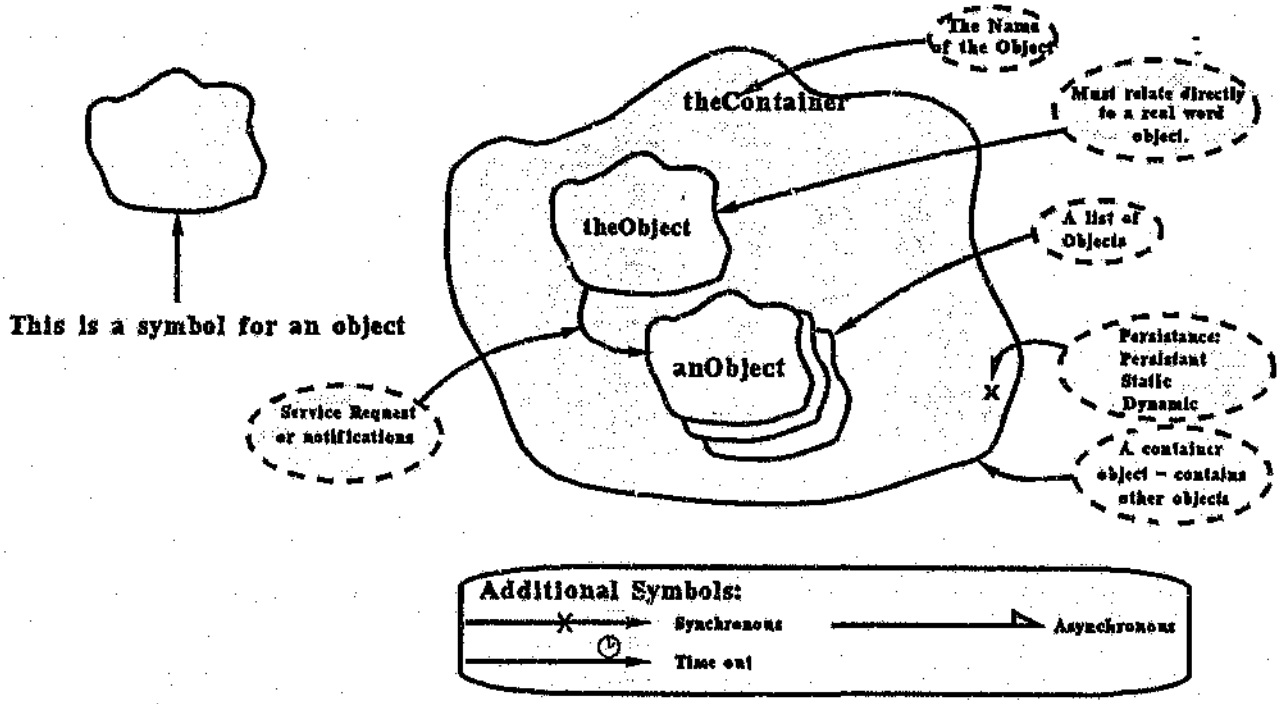


Figure 3.10: The Object Diagram (proposed by Booch[45])

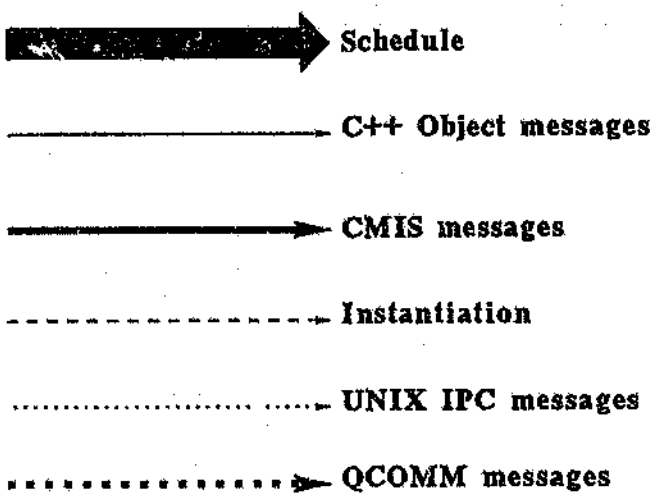


Figure 3.11: The Object Diagram Extensions for Modelling

Investigating the OOA/OOD methodologies for this study revealed that there is a wealth of literature in this field. For the purposes of the project and study, a modified but limited set of the methodologies were practised.

I am currently investigating richer and more comprehensive OOA/OOD/OOP environments for future development phases of this project. Of particular interest are CASE environments which are now becoming available as mature products applicable to the object-oriented paradigm. At time of writing, the TeamWork products from CADRE Technologies were being investigated.

3.9 CHAPTER SUMMARY

After the motivation for an object-oriented component approach in the previous chapter, this chapter introduced the object-oriented paradigm. Some of the more important considerations relating to the adoption of the new paradigm were also reviewed. (Review of this material has been necessary before attempting to leap directly into the design of the core components).

The current available software and hardware technologies relevant to this project were described, since this has a impact on the degree of re-usability, portability and system performance. As observed, the faster hardware available these days and the formulation of standards has a positive impact on the effectiveness of the object-oriented approach.

However, standards bring with it penalties in terms of more generic, heavier and complex software layers - which compromises the performance and efficiency, particularly in real-time systems.

This chapter also described the object-oriented life-cycle that was employed and refined for this project, and reviewed the major differences between this life-cycle and that of the traditional approach. Closely associated with the object-oriented life-cycle are the roles of application consulting, application engineering and component engineering, and the model used on this project was described.

This chapter also reviewed some of the 'soft' issues that had to be considered at the start of the project, particularly whether object-orientation was the right methodology and technology to use for the technical real-time environment. Finally the project management and control mechanisms used on this project were reviewed, as well as the object-oriented analysis, design and documentation methodologies employed.

With the motivation and introduction of the object-oriented component-based approach complete, the stage has been set for the design and implementation of the set of generic components for real-time systems.

4. DESIGN AND IMPLEMENTATION OF THE CORE BUILDING BLOCKS

Up until this chapter, the object-oriented component-based approach to building real-time systems has been introduced and motivated. Several important issues relating to the adoption of this approach have also been reviewed in the previous chapter.

This chapter introduces the five basic subsystems characteristic of most technical real-time systems, and then proceeds to describe the 3-layer structure model that was used to design the generic components of these subsystems. However, to better understand the structure and design of the various subsystems, the software bus and the generic requirements and architecture of technical real-time systems are first discussed.

The components generic to of real-time systems are detailed in this chapter, and the following chapter describes the higher-level TNM domain-specific components. The Appendices contain examples of the design, modelling and implementation of some of the components.

4.1 THE GENERIC ARCHITECTURE OF REAL-TIME SYSTEMS

This section reviews the architecture and generic requirements of technical real-time systems relevant to this study. The purpose of this review is to enhance the reader's understandability of the design of the object-oriented components described later in this report.

Generally in distributed real-time systems (refer to Chapter 1 for the definition of real-time systems relevant to this study), there are many software processes running asynchronously on inter-connected computers. Some of these processes are performing interfacing to the world being monitored and controlled, other processes are involved with processing of changes, logging of events and alarms, and performing controls on the outside world. Other processes perform configuration, archiving, and man-machine interaction - and all processes are communicating and synchronising with one another in real-time. That is, the time between any events occurring (eg. outside events and operator initiated) and their associated responses must be deterministic in time.

By analysing the architecture and functionality of our organisation's previously developed real-time applications, a generic software architecture was identified that could be implemented with the object-oriented methodology. In keeping with object-orientation, it was important that the major components of this architecture were defined not only along functional lines, but they also had be independent of each other's data structures.

Six major subsystems or components were identified:

(1) the communications infrastructure
(2) interface to the world (front-end)
(3) man-machine interface
(4) real-time image and processing

(5) configurator with configuration database
(6) historian with historian database

These subsystems and the information-flow relationships are depicted in Figure 4.1 below:

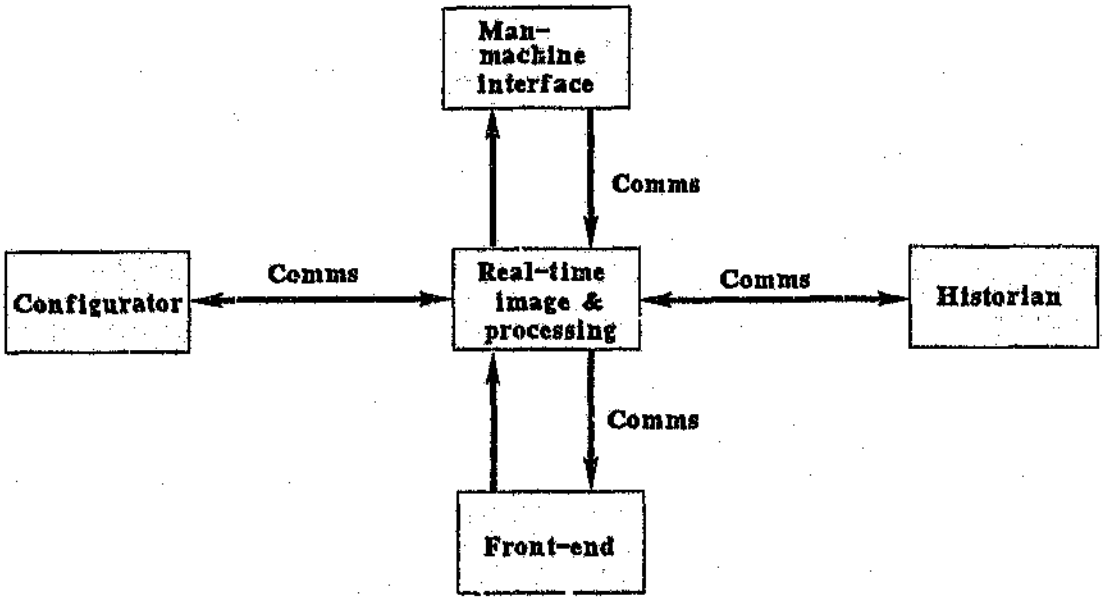


Figure 4.1: Generic Architecture of a Real-time System

Defining these distinct subsystems as above lends itself to the whole object-oriented paradigm. Each of the subsystems is a grouping or super-set of further object-based components and sub-components, and components within each component subsystem are related in terms of the required behaviour and functionality. These generic subsystems are described below.

Communications subsystem = Software Highway, since it was desired to have a common highway or bus to which any application processes could attach, and talk a standard protocol to any other processes, much like a computer hardware bus.

Front-end = World-Machine Interface (WMI), since it is this subsystem that provides the interface between the real-time system and the outside 'world' that is being monitored and controlled.

Man-machine interface = Man-Machine Interface (MMI). This provides the interface between the real-time computer system and the users.

Real-time database and processor = Real-Time Object Management System (RTOMS). This performs all processing, calculations, derivations and real-time storage/image of the objects that model the real world being monitored and controlled.

Configurator with configuration database = Configurator. This component deals with the storage and management of all configuration data pertaining to the real-world objects being monitored and controlled. This includes the configuration which defines the mapping between real-world physical objects and logical objects, as well as the relationships between the objects and definition of object hierarchies.

Historian with historian database = Historian. This component performs the storage, retrieval and management of all medium and long-term information, typically logs, errors, alarms, and events pertaining to the real-world equipment being monitored and controlled.

For the remainder of this report, these generic subsystems are referred to by their proprietary names as described above.

With the generic architecture outlined and major subsystems identified, it was then possible to design the structure of the components of these generic subsystems.

4.2 THE STRUCTURE OF THE CORE BUILDING BLOCKS

In this section, it is intended to outline the overall approach adopted for designing the structure of the building blocks or components. It is shown how the component structure forms the foundation of the object-oriented approach and determines the approach's success, particularly in terms of component re-usability, testability and maintainability.

In keeping with the object-oriented component-based approach to building real-time systems, a unified effort was made to structure, abstract, and layer the design of the subsystems to practically maximise the generality and re-usability of the components.

A particularly difficult challenge was to design the philosophy and architecture to be as generic as possible, but within certain constraints. It was required to have a core component set that would be applicable to most technical real-time systems. Our aim was an ideal 60% level of re-use into new application domains. (The target application domains were primarily telecommunications network management and industrial supervisory and control).

However, it was important not to implement a core component set that was over-generic. This 'solution for all possible problems' approach would suffer from inflexibility, non-ideal fit to requirement, lack of performance, and of course the cost effort of implementing such a core component set.

An important factor in the design of the generic components was to design components and component frameworks into intermediate abstraction levels. (The framework is the term used by recognised authors for a collection or abstraction of object/class components that are specific to an application, and represent the largest granularity for re-use). This is convincingly proposed by Fischer[24] and Johnson[37] since this strategy allows easy re-use by recombining existing intermediate abstractions with other existing abstractions and new abstractions to form new applications and systems.

Initially it appeared that a basic 2-layer approach was suitable. This approach is implied by Gibbs et al[27], who advocates a generic set of 'boilerplate' components, with only a few software components being designed to meet the requirements of the specific application. That is, the bottom layer of components generic to most real-time distributed systems, and the upper layer which would be very application specific. This is depicted in Figure 4.2 below.

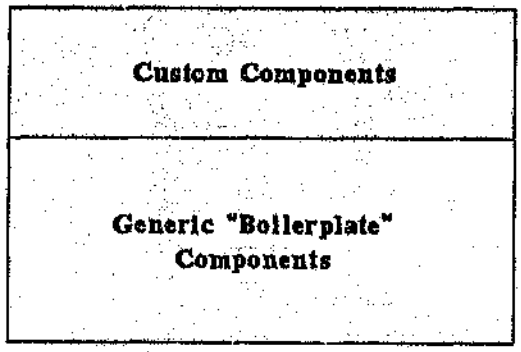


Figure 4.2: The 2-Layer Building Block Structure

As Gibbs et al[27] observe, a higher level of re-use is achievable if the application domain is well characterised. In application domains such as telecommunications network management, industrial process monitoring, and electricity reticulation management, I found that there existed strong characterisations.

To prove the point, the initial MMI prototype development demonstrated that the granularity of the component structuring was too course. It became evident that the upper layer customised portion constituted 40 to 50% of the total software.

For these reasons it was apparent that an intermediate layer of components characteristic to the application domain (in this case TNM) would improve re-usability. And so was born the 3-layer philosophy, with a new intermediate layer of components that would consist of components generic and re-usable to other applications in the same application domain. I believed that for application domains such as TNM, industrial process monitoring, electricity reticulation and others, a distinct set of re-usable components could be abstracted characteristic to that application domain, resulting in a potentially far higher level of re-use in other applications of the same domain.

This study is contained to the design and development of an application in the telecommunications network management domain. In this study, these 3 basic layers of components have been termed *ObjectView*, which is the lower level generic to most

real-time systems, *AccessView*, the domain-specific components relevant to the telecommunications network management domain, and *CustomView*, which is the application-specific customised components.

I have chosen to depict this structure as a diagram consisting of layered and inter-related set of building blocks. This is shown in Figure 4.3 below:

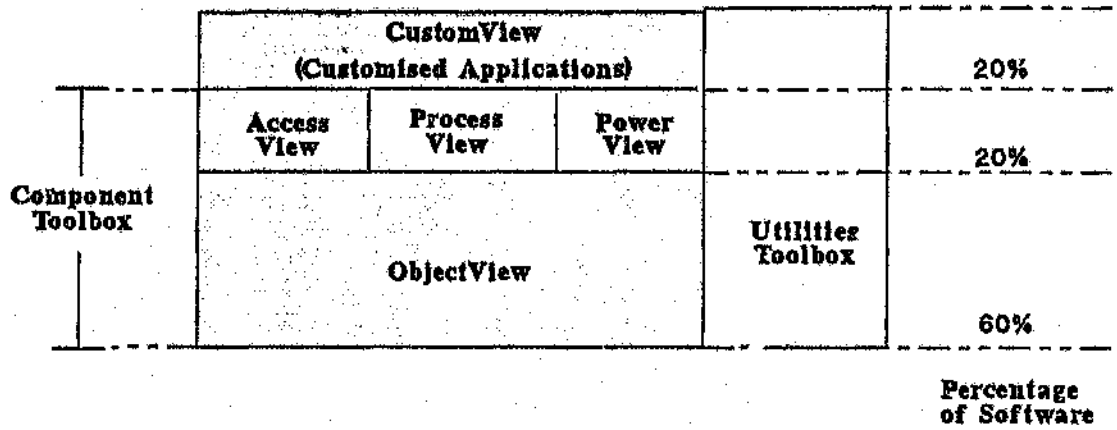


Figure 4.3: The 3-Layer Building Block Structure

Proceeding from this point, each of the generic subsystems, that is MMI, WMI, RTOMS, Historian and Configurator were then divided into smaller abstracted components or frameworks (Johnson[37], Fischer[24]) structured into this 3-layered model. For example, the MMI subsystem was designed as a set of structured and related components, some of which fall within the generic *ObjectView* layer, others in the domain-specific *AccessView* layer, and the remainder in the *CustomView* layer.

Following below is a brief description of each of the layers referred to in Figure 4.3 above.

ObjectView

ObjectView was the name given to the core generic object-oriented components that was envisaged composing at least 60% of the total software in new real-time applications. (Note that *ObjectView* was the name originally given at the early stages of this project, and should not be confused with the Borland's OBJECTVIEW product). It would include the generic software related to inter-process communications, database facilities, man-machine interfacing, front-end interfacing and real-time process management.

AccessView

This was the name given to the re-usable components generic to applications in the telecommunications network management (TNM) domain. It was based, and built on the *ObjectView* layer - and can be viewed as an extension of the *ObjectView* objects (and object subsystems) with a telecommunications network management 'flavour'. It incorporates the modelled objects required for generic network management functions, such as monitoring, routing and reporting of network alarms, and issuing controls to the telecommunications network.

It was envisaged that this building block would make up at least 20% of new telecommunications network management application software. The functions of the *AccessView* building block are described in more detail in following chapters.

ProcessView and PowerView

These layers are similar to *AccessView* in philosophy, except that *ProcessView* is an extension of the *ObjectView* components incorporating the modelled functionality required for applications in the industrial process monitoring and control domain. Likewise *PowerView* is the building block for real-time electricity reticulation applications.

The *PowerView* and *ProcessView* building blocks have been shown here for illustrative purposes only, and are not part of this study. The scope of this study was to design and implement only *AccessView* (Telecommunications Network Management) of this domain layer. Figure 4.3 illustrates how other domain specific building blocks can be built on top of the generic *ObjectView* layer.

Customised Applications

This is the application-specific layer of software that was built and based on the underlying domain-generic building blocks and their sub-components. It was envisaged that this would make up the remaining 20% of software. It is this layer which is the fully customised portion of a new system, and is specific to an actual application ie. it will not be generic to another application, even in the same application domain.

The Utility Toolbox

This was designed as the toolbox of re-usable entities and functions that may be used by any of the other components. They include generic utilities such as error reporting, tracing, logging, command tracking, watchdogs and object-manager housekeeping functions.

It also contains utilities that are not object-oriented in terms of implementation. This was a particularly important requirement since our organisation has a substantial investment in generic utilities and function libraries previously written in conventional C. Since it is major task to up front rewrite all these utilities using object-orientation and C++, it was necessary to be able to have access to these utilities and libraries from the object-oriented environment.

The Component Toolbox

The Component Toolbox was the general name given to the entire set of generic re-usable components for distributed real-time systems, ie. the real-time environment generic *ObjectView* components, as well as domain-specific components such as *AccessView*.

In this section the layered building block philosophy was introduced, and use of a 3-layered building block structure for this study was motivated. The basic structure of the core building blocks was described and at the same time certain terms and proprietary names that are referred to later in this study have been defined.

Later in this report, it will be evident how important this general 3-layered building block structure is. It will be shown how this forms the foundation of the more detailed component designs of the real-time software subsystems. It will also be demonstrated how this 3-layered building block philosophy does maximise the re-usability, testability and maintainability of the object-oriented approach to building a real-time TNM system.

4.3 THE SOFTWARE HIGHWAY

Apart from adopting the object-oriented methodology for this project, the next biggest challenge was deciding on, and designing the communications philosophy. This was viewed as the heart of the system, since the entire system is dependent on it not only for communications, but also for overall system performance and integrity.

The following sections reviews the conventional communications approach and motivates the need for a software bus communications philosophy, with emphasis on its importance to the overall success of the object-oriented approach.

Basic details on the software highway options and design are also described. However further technical information on the software bus and its CMIS-conformant implementation is contained in Appendix C.

4.3.1 The Conventional Communications Approach

To motivate the need for a 'software bus' communications philosophy, and emphasise its importance to the overall success of the object-oriented approach, the conventional method of inter-process communications (used by our organisation) in real-time systems is first reviewed.

Our organisation's approach has been based on the communications/messaging facilities provided by the underlying operating system and platform. In the case of UNIX this would be UNIX messages or TCP/IP or BSD socket communications. Communications between processes on different nodes on a network would consist of processes sending asynchronous messages to the addressed process on the addressed node. This is depicted in Figure 4.4 below:

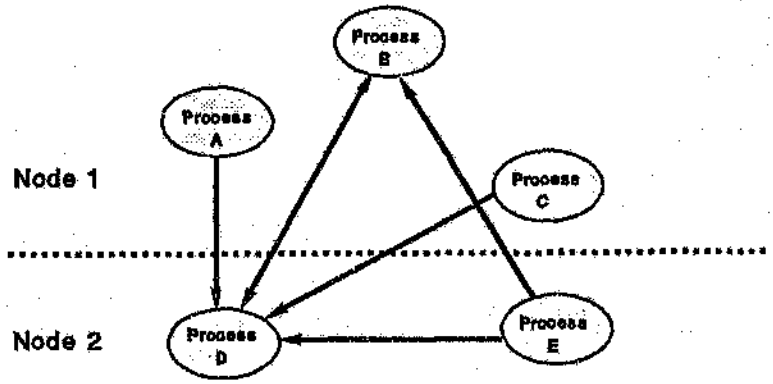


Figure 4.4: Conventional Inter-process Communications

The above approach, although very efficient, has several drawbacks:

* It is fixed, inflexible and very customised.
* Every process is required to know the address i.d. of the destination process, and the destination process node i.d.
* Any changes to the message datagram structure or format, or change of location of a process results in code change. This inflexibility seriously affects non-embedded real-time systems.
* It is generally dependent on the operating system and to a certain extent the hardware.
* Any communications software written on top of the underlying operating system messaging facilities makes the communications system extremely proprietary in terms of data structure, syntax and protocol, and therefore is not readily open to other systems. In these cases special interface or gateway software has to be implemented.
* It is not conformant to any formal standards, and with the move to Open Systems, it is becoming more and more difficult to motivate to the end-user use of proprietary communications subsystems and protocols.

For the above mentioned issues it became clear that the conventional approach was not suited to the proposed component approach to building real-time distributed systems, where we desire communications flexibility, transparency and standardisation. These attributes are necessary to enhance the re-usability of the communications subsystem itself as well as the components which utilise the communications infrastructure.

4.3.2 The Requirements of a Software Highway

Having highlighted the issues associated with the conventional inter-process communications approach in the previous section, what were the specific requirements for communications in the object-oriented approach?

* It had to be conformant to industry standards. This would ensure inter-operability, system scalability, extensibility and flexibility.
* Performance had to be acceptable for use in real-time distributed systems.
* It had to provide transparency to the physical location of the inter-communicating processes.
* Changes to the lower communications stacks should not affect application processes, and changes to application processes should not require changes to the communications subsystem. In other words, a standard application programmer interface (API) was necessary.
* It had to lend itself to the object-oriented paradigm, with objects communicating with objects.
* The message content and syntax definition should be contained in the message itself, and not be defined in separate include files.

At the same time as my original experimentation and prototyping with object-orientation, I began researching OSI's reference model and the proposed OSI Common Management Information Services and Protocols (CMIS/CMIP). (Further detail on this is contained in Appendix C of this report and extensive references are given in the bibliography).

The OSI CMIS/CMIP model was one example that provided the concept of a Software Bus and satisfied many of the requirements desired for the object-oriented approach to building distributed real-time systems, particularly standardisation.

With the generic real-time systems architecture described in an earlier section, and the software bus proposed here, the architecture of a real-time system could be represented as in Figure 4.5 below.

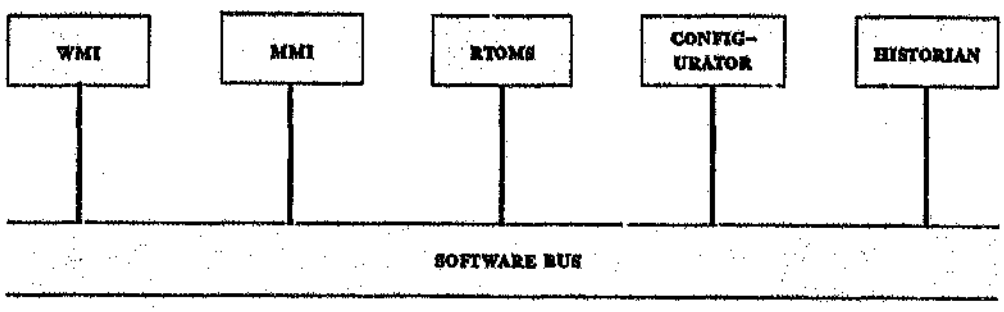


Figure 4.5: Generic Architecture of a Real-time System with Software Bus

4.3.3 The Approach Adopted for this Project

The preceding sections have reviewed some of the expected problems and issues associated with using the conventional communications approach in object-oriented real-time systems. Ideal requirements for a communications approach suited to the object-oriented paradigm were also outlined. This section describes the options that were available and presents the arguments for the various options. Finally the CMIP-conformant Software Bus approach is motivated as being the better solution for the project.

Having researched the current standards proposed by OSI for Open Systems (refer to the section in Chapter 3 on standards) and keeping a general watch on industry trends and end-user attitude and commitment to these standards, there were several communications options which could be adopted for the object-oriented approach, namely:

(a) Continued use of our organisation's proprietary communications approach.
(b) Use the already well-accepted SNMP (Simple Network Management Protocol) which is based on the TCP/IP communications protocol stack.
(c) Communications based on the OSI 7-layer stack and using OSI's defined CMIP (Common Management Information Protocol) protocol.
(d) Communications based on the TCP/IP communications stack and using an upper-level protocol which is OSI/CMIP-conformant. This is known as CMOT (Common Management Information Over TCP/IP).
(e) Implementing a CMIP/S-conformant API layer on top of our organisation's proprietary communications subsystem (of option(a)).

Option (a) was rejected for reasons already outlined. Further evaluation of the above options showed that option (b), SNMP, although popular, was static and was perceived by industry as being on its way out, being gradually phased out in favour of OSI's CMIP (see bibliography (1,2,4)). Option (d) is not widely used and seems unable to gain acceptance (see bibliography (1,2,14)). It was clear that certainly the network management industry was pushing toward option (c), OSI's CMIP protocol (see CCITT Study Group XV(61) and relevant bibliography references).

The problem is that there is much talk and discussion about OSI communications stacks, but there is minimal implementation. Particularly in South Africa, it appears that everyone is watching everyone else to see who takes the first big steps to converting their large networks to the OSI standards for telecommunications networks management.

A second problem is that the OSI standards for CMIP and CMIS have not yet fully matured and there are few products available that provide full working conformance. For most part these standards still exists as proposals and definitions.

The last option (d) involved implementing our own communications layer which would provide CMIS-conformance. This layer would be written on top of our existing proprietary TCP/IP-based communications subsystem.

Our organisation already has an in-house developed proprietary inter-process/cross-nodal communications subsystem based on TCP/IP. This product, called QCOMM, offers transparent process-to-process communications across UNIX, RTE (Hewlett-Packard) and DEC/VMS platforms.

It also provides message buffering, prioritisation and management of system resources

used by the communications system, an essential requirement for realtime systems. This product has thus far formed the communications backbone of all systems delivered by our organisation and is stable and reliable.

The QCOMM product already provided communications up to OSI's defined Presentation Layer. By providing additional communications software on top of QCOMM as well as an Application Programmer Interface (API), most of the required CMIS services as defined by OSI could be provided. At the same time this approach provides for upgrading of the QCOMM/TCP communications stack to OSI at a later time. Because of the programmer API and CMIP/S conformance, there would be little impact on current applications when a new communications stack is used.

The main question asked was why develop our own communications subsystem when conformant OSI-CMIS communications stacks were indeed available, such as from RETIX and Hewlett-Packard. The reasons are summarised below:

(a) Network management standards are not yet mature and stabilised, particularly those based on OSI's CMIP.
(b) There is already extensive use and investment in TCP/IP networks, and many end-users are sceptical about CMIPs benefits and are unwilling to wait for OSI-based network management solutions.
(c) For this project, our organisation required a short time to market for competitive reasons.
(d) The costs involved of the OSI-CMIP route: the cost of evaluation, the high cost of the OSI stack products, and the costs of integration, testing and optimisation of software not really under our control.
(e) There existed a growth path - at any later time the TCP communications stack could be upgraded to an OSI communications stack without affecting current applications.

So the above reasons provided enough motivation for implementing our own communications API layer which would provide CMIP/S-conformance. This layer would be written on top of our organisations proprietary QCOMM communications subsystem.

This section has motivated the use of our existing in-house inter-process communications subsystem, but with an added layer on top to provide the CMIS services as defined by OSI. From an application point of view, this would provide CMIS service conformance, and the underlying QCOMM would serve only as the transport mechanism for the messages. This approach satisfies the object-oriented communications requirements identified earlier. What remained to be done was to prototype and implement this additional software and evaluate. Of particular concern was the performance of such a communications subsystem, because CMIS conformance implied high generality resulting in complex, multi-layered software.

4.3.4 Developing the Communications API

The additional communications software built on top of QCOMM consisted of software to provide the CMIS conformant services, as well as a standard Application Programmer Interface (API). This section follows on from the previous sections in this chapter, and briefly describes the programmer interface which the project team developed.

SOMI was the proprietary name given to the API, an acronym for Standard Object Management Interface. Thorough study and investigation of the CMIP/CMIS specifications was done and a full specification was drawn up before this API was implemented. Figure 4.6 below shows how the SOMI layer fits in the network model, and its relationship to the underlying communications layers and protocols.

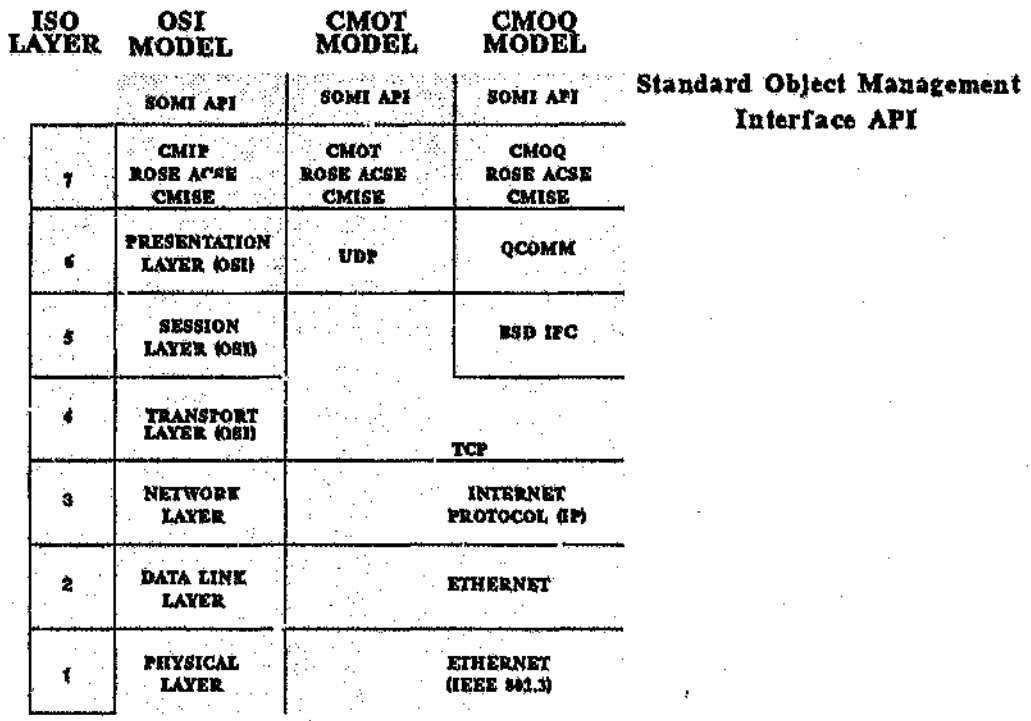


Figure 4.6: The Communications Network Model

The above figure depicts the SOMI API layer, on either the CMIP, CMOT or CMOQ communications stacks. (CMOQ is not an industry standard acronym - it was termed by the project team, an acronym for Common Management Information Over QCOMM) The figure depicts how the same SOMI API could be built on top of the OSI/CMIP stack or the OSI/CMOT stack (Common Management over TCP/IP).

In designing and implementing SOMI, several design and philosophy parameters were defined:

* SOMI had to be conformant to OSI's CMIP/CMIS to provide communications to other existing and future OSI network entities and elements.
* It had to have an easy to use API.
* It had to have a clean interface on the underside ie. to QCOMM, since this would

allow replacement of QCOMM with a full OSI communication stack as and when required in the future. This migration from TCP/IP to OSI must have no effect on existing applications.

* In design and implementation, performance optimisation was a serious requirement, since the project was a real-time application.

The SOMI software was developed by the project team, and took only 6 man-months to develop and fully test. This included near-full conformance to OSI's CMIS definitions. Because of the generality and flexibility defined by CMIS, the API was initially cumbersome and complex, and several prototyping iterations were required to obtain a satisfactory API. Surprisingly, the eventual performance results were quite satisfactory. Further details on performance are contained in Chapter 6, "Analysis of Approach", and further technical detail on aspects of CMIS, CMOQ, SOMI and the SOMI API are contained in Appendix C.

4.3.5 Object-Managers and Applications

There exist special names for two types of processes or programs that are associated with CMIP/S and object-orientation. These processes or programs are called *Object-managers* and *Applications*, and are briefly explained here, since there are several references to these later in this report.

Object-managers can be likened to 'Server' or 'Agent' processes which understand incoming requests from remote or local 'Client' processes. *Applications* can be likened to these 'Client' or 'Manager' processes from which the requests originate.

In terms of CMIS however, the major difference between *Object-managers* (OMs) and *Applications* is that the *Object-manager* contains 'Managed Objects' (MOs), and *Applications* do not. With an *Object-manager*, all MOs within its control are registered within the *Object-manager's* containment tree (also described as a registry or directory). This means that any *Object-managers* or *Applications* may address requests at another *Object-manager's* objects via the containment tree.

However, *Applications* do not have Managed Objects under their control, and can therefore receive no external requests to its objects. *Applications* can only initiate requests. A good example of a CMIS *Application* is a report program which would request attribute data from objects in the Historian *Object-manager* (which would be a CMIS *Object-manager*) and display the data in a formatted report.

Both *Object-managers* and *Applications* have to attach themselves to the Software Bus before they can service or initiate any operations. It should be noted that *Applications* are implemented in an object-oriented manner with classes and objects, but because it has no CMIS containment tree, its objects are internal only and are not externally visible to other *Applications* or *Object-managers*.

Further detail on SOMI and *Object-managers* and *Applications* are contained in Appendix C. This Appendix also contains pseudocode examples of the *Object-manager* and *Application* frames.

4.3.6 In Summary

The chapter on the Software Bus has reviewed inter-process communications issues and motivated reasons for implementing a CMIP-conformant API based on our organisation's existing TCP/IP-based QCOMM communications product.

It has also identified the principal requirement that the API should offer an easy migration path to an OSI stack when required in the future, without affecting the existing applications. The reader will note that much attention has been given in this chapter on the 3-layered building block approach and the CMIS/CMIP software bus. The reason for this is that it is these two systems concepts that have played a major contribution to the success of the object-oriented approach in building real-time systems. This conclusion will become further evident in the later chapters of this study, particularly in the analysis of the object-oriented approach.

Once the 3-layered building block philosophy had been defined, and the design of the SOMI software bus completed, the detailed definition and implementation of the real-time subsystems could take place. In the subsequent sections, each of the subsystems' structure, design and implementation is described.

4.4 THE MAN-MACHINE INTERFACE

In this section, the Man-machine interface is described. The generic requirements for the *ObjectView* MMI subsystem is defined as well as the building block structure of this subsystem. This section also demonstrates the potentially high level of component re-usability in the MMI subsystem.

In keeping with the 3-layered structure of the building blocks previously described, for each of the subsystem components, Chapter 4 will concentrate on the *ObjectView* core components generic to distributed real-time systems. Chapter 5 deals with the application-domain and application specific components (*AccessView* and *CustomView*) in more detail.

For the sake of readability and relevance to this study, specific details on the object-oriented design, modelling and implementation of the MMI components have been omitted.

4.4.1 The Generic Requirements of the MMI

For the purpose of developing *ObjectView* components which would be generic to most technical real-time systems, the following basic requirements were identified for the MMI subsystem:

* It had to provide an X-based graphical user interface, based on the X/Motif standards.
* It had to also provide for a terminal-based text user interface. The reason for this is that there is already a large installed base of non-graphics terminals and for reasons of cost it is not always possible to upgrade these to graphics terminals or workstations.
* Components (in the form of component libraries) for displaying and manipulating

text and graphics symbols on graphical and character user-interface screens.

* Picture Editor and Mimic Editor components, which would enable a user to interactively define new picture elements, and create and maintain mimic diagrams.
* The Mimic Editor to provide for definition of live 'dynamic links', linking screen symbols to real-time statuses of attributes of the defined objects in the Real-Time Object Management System (RTOMS).
* The mimic display subsystem to include features enabling automatic picture/symbol composition on the mimics dependent on the current configuration as defined in the network's central configuration database at the time of display call-up.
* The different MMI components to be structured with clean interfaces to facilitate selective component re-use in other applications.
* To provide for context-sensitive help at *ObjectView*, *AccessView* and *CustomView* component levels.
* To provide security and protection to prevent unauthorised operator violation of access and control.
* To provide the necessary generic housekeeping functions and utilities including startup, shutdown and system statistics.
* To provide the mechanisms for driving an application-specific user interface via three modes or windows: a graphical view, a textual view, and a detailed view with statistics and logs. It must be possible to navigate between these modes or views on a context-sensitive basis.
* The core MMI *ObjectView* components to have no domain or application specific functionality.

With the above basic generic requirements defined, the component structure of the MMI subsystem was designed, and this is described in the following section.

4.4.2 The MMI Building Block Structure

With the generic requirements outlined in the preceding section, the required MMI building blocks or components were designed and implemented. The structure and layering of these components is depicted in Figure 4.7 below.

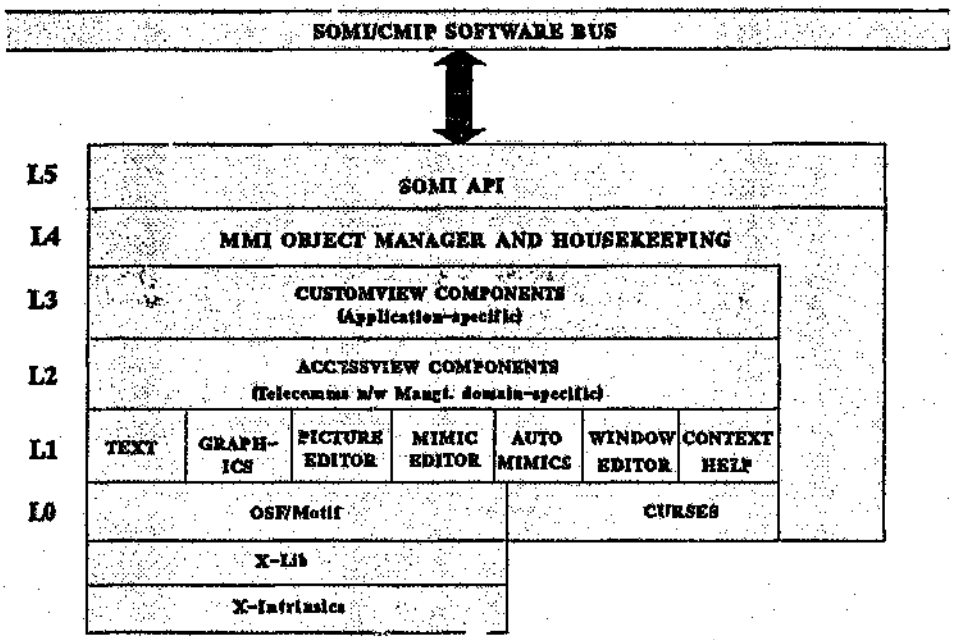


Figure 4.7: MMI Subsystem Building Block Structure

The Levels L0 to L5 referred to in Figure 4.7 above are described below.

Level 0 (L0) These are the libraries that were used as the foundation of the MMI building blocks. For the graphical user interface, the Motif/X libraries were used, and for the Character User Interface (CUI), the UNIX-standard CURSES libraries were used. As Guimaraes[30] proposes an extension of Borkin's[14] database modelling terminology, this level provides the syntactic level tools.

On the graphical side, the most basic access is provided by the Xlib library of primitive calls to the X-Window. However programming with this level of abstraction is tedious and requires a great amount of attention to detail. Motif provides a higher level of access together with a window management system through 'widgets'. Although the Motif Toolkit does provide extensive libraries for programming graphical user interfaces, and is object-oriented, it unfortunately does not provide interfacing via the Object-Oriented Programming (OOP) C++ constructs directly.

In terms of the character user-interface, the use of the CURSES libraries

provided by UNIX offers many advantages including terminal-device independence. However, these libraries are not object-oriented and are also tedious to work with.

Level 1 (L1) This level is the *ObjectView* generic level of components for real-time systems. This level provides the semantic level of components (Guimaraes[30]) that are closely related with the abstract and generic to technical real-time systems. The components provided include the object-classes and associated methods for displaying and manipulating text and graphics displays, the managing of pull-down and pop-up menus, scroll-bars and multiple window arrangement.

Level 2 (L2) This level represents the domain-specific generic components. This is the level where the semantic level of components associated with the telecommunications network management domain (*AccessView*) will be provided. Further details on the *AccessView* components are contained in Chapter 5.

Level 3 (L3) Application-specific components. This is the level of components that require customisation for each particular TNM application. These embody semantics very specific to the particular application being developed. Further details on these *CustomView* components are also contained in Chapter 5.

Level 4 (L4) This level embodies the generic *Object-manager* and the associated housekeeping functions. The *Object-manager* has been described earlier in this Chapter, and further detail is contained in the Appendix on the Software Bus. This is a generic 'program frame' that holds the object-containment tree and functionality required for SOMI as well as a 'program frame' for embodying the lower level object-oriented components. This component can be regarded as part of *ObjectView* since this 'program frame' or template is generic to any object-manager process.

Level 5 (L5) This is the SOMI Software bus Application Programmer Interface - the actual function calls providing access to the SOMI Bus. This *ObjectView* component is closely associated with the *Object-manager* component.

The component building blocks as described above were implemented as component class libraries, with the exception of the MMI *Object-manager* which is the process that integrates the objects from the required lower-level components .

Each component was built on top of lower layer components, with extensive use of multiple inheritance. The multiple inheritance was specifically directed at creating 'abstracted' objects that inherited from separate classes defining the interface and implementation. Therefore if implementation is changed, the virtual abstracted object (that used by the application) is unaffected. Change is only required to the implementation class. We found that multiple inheritance provided better re-use and a more generic set of components, especially with real-time technical systems which are highly dependent on external environment details. It was also found that the MMI subsystem was particularly suited to component re-use because there is much software behind the visual user interface that is generic to technical real-time applications.

The MMI components as described above were designed with clear interfaces between the different layers. That is, the programming interfaces between the different horizontal layers were clearly defined, with the objective of providing a 'mix and match' type of component re-use.

A typical example of this is the TEXT component at level L1 which could utilise either the OSF/MOTIF component at the lower level L0 (for graphical user consoles), or the CURSES component for character consoles. The TEXT component would not be aware of which components it was interfacing to at the component levels below or above. It will be observed that this 'mix and match' philosophy was characteristic of the designs of the other subsystems too.

This section has described the design of the MMI subsystem's bottom level (ie. the *ObjectView* level) of the 3-layer philosophy described earlier. The second and third layers, that is the domain-specific components and the customised components, are described in Chapter 5.

4.5 THE WORLD-MACHINE INTERFACE

The World-Machine Interface (WMI) is described in this section. The WMI provides the interface between the outside real-world being monitored/controlled and the computer system.

It was observed that the WMI did not have as high level of general re-use potential as the MMI subsystem. The reason for this is that the interfacing to the outside world is very specific to the type of equipment being interfaced to, and the type of equipment is specific to the application domain.

However, this section does describe the structure and design of those core WMI components that were considered to be generic to most real-time application domains - the *ObjectView* components.

Again, for the sake of readability and relevance, specific technical details on the design, modelling and implementation of the WMI components have been omitted.

4.5.1 The Generic Requirements of the WMI

For the WMI, the following basic requirements were identified for the generic *ObjectView* components:

* It was to behave merely as the interface between the outside world equipment and the RTOMS subsystem, and no real-time *processing* was to be included in the WMI functionality.
* It was to be structured in a way such that new 'device-drivers' and 'interface-drivers' could be added in a mix-and-match manner.
* It had to provide for polling, receipt of asynchronous data, sending controls and configuration downloads.
* It had to provide Q1, Q2 and Q3 levels of interface to the outside world, where Q1 and Q2 are lower level protocols as yet not fully defined by CCITT's G.773 recommendation, and Q3 is the CMIP interface as adopted by IEEE for Network

Management. (Refer to Appendix A and the CCITT Study Group XV reference[16] for further details).

The design of the WMI component structure was then based on the above basic requirements and the details are contained in the following section.

4.5.2 The WMI Building Block Structure

The design of the WMI component structure is depicted in Figure 4.8 below.

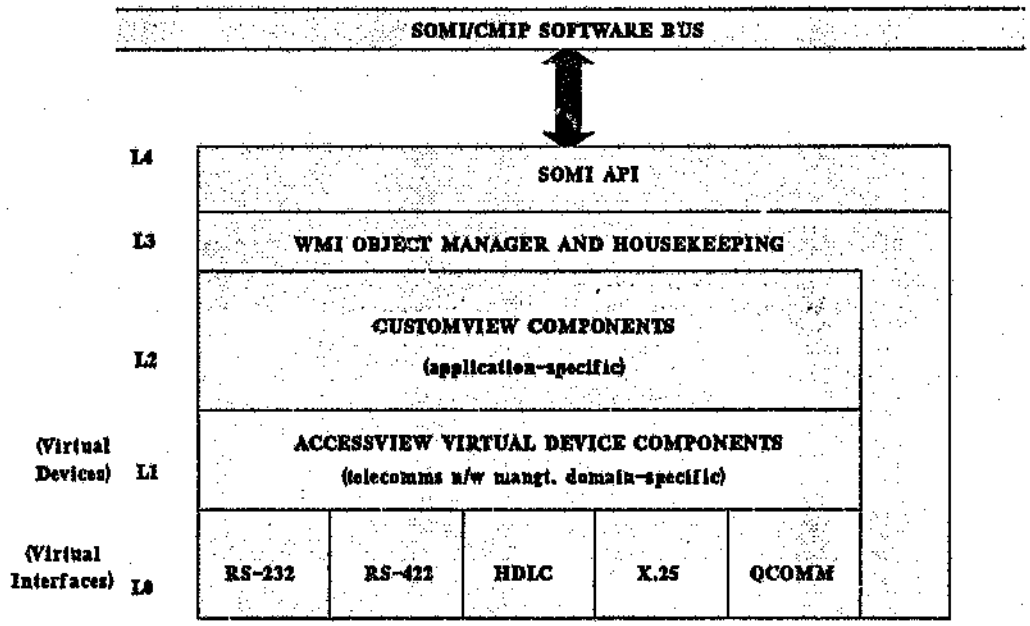


Figure 4.8: WMI Subsystem Building Block Structure

The Levels L0 to L4 referred to in Figure 4.8 above are described below.

Level 0 (L0) These low-level components provide the object-oriented data-link level interface to the real-world equipment. They are referred to as virtual interfaces, and for this project included interface drivers for RS-232, RS-422, HDLC, X.25 and QCOMM. The QCOMM virtual interface was required for interfacing to our organisation's existing TNM networks utilising the basic proprietary QCOMM messaging subsystem. In other words, this virtual interface behaves as a 'gateway' device.

At the upper end, these virtual interface components interface to the Virtual Devices via a defined function call APIs. This implies that any Virtual Device can be matched to whatever Virtual Interface is required for physically interfacing to the real-world devices.

Level 1 (L1) This level consists of the Virtual Devices that are specific to the *AccessView* application domain. Each Virtual Device component embodies the functionality and behaviour required to poll, drive and control the real-world equipment it interfaces to.

As with the Virtual Interface components, the upper end of the Virtual Device components interfaces with the *CustomView* components via a defined standard function call API.

For this project, these Virtual Device components consisted of *AccessView* components for the telecommunications network management domain. These *AccessView* components are described in detail in Chapter 5.

Level 2 (L2) This level contains the customised *CustomView* application-specific components. They typically contain the polling and state algorithms specific to the application. Additional detail on these components is contained in Chapter 5.

Level 3 (L3) This is the generic object manager with the associated housekeeping functions. It is essentially a generic program or process 'frame' for linking together the lower-level object-oriented components

Level 4 (L4) The SOMl Software Bus Application Programmers Interface - the actual function calls providing access to the SOMl Bus. This *ObjectView* component is closely associated with the *Object-manager* component.

As with the MMI subsystem, each layer was built on top of the layers below, with each component being contained and abstracted so that re-use, maintainability and testability are improved.

From the design and implementation of these WMI components, it was observed that the generic *ObjectView* components, that is the Virtual Interfaces, were very thin in this subsystem. However, the *AccessView* domain-specific Virtual Device components of Level 1 proved to be quite substantial.

4.6 THE REAL-TIME OBJECT MANAGEMENT SYSTEM

In this section the Real-Time Object Management System (RTOMS) is described. It was observed that of all the subsystems, the RTOMS had the lowest level of potential re-use at the *ObjectView* level. That is, the components in this subsystem were very specific to the telecommunications network management domain and the application. It was observed that the low level of re-use was due to the fact that the RTOMS contained much of the functional models specific to TNM applications. Because of RTOMS' close relation to the application domain, it was found that the *AccessView* components were quite substantial, and this is described in more detail in the RTOM's *AccessView* description in Chapter 5.

This chapter reviews the generic requirements of the RTOMS *ObjectView* components, and details the component structure that was designed for RTOMS. Further technical details and examples of the object-oriented design, modelling and implementation of the RTOMS is contained in Appendix D.

The details of RTOMS have been included in an Appendix for two reasons: to better convey the RTOMS model, which I considered to be the most important subsystem, and secondly to demonstrate an example of the object-oriented design, notation and programming methodology used on the project.

4.6.1 The Generic Requirements of the RTOMS

For readers not familiar with the requirements and concepts of real-time databases, the following paragraph reviews the RTOMS philosophy.

RTOMS Philosophy

The RTOMS is a memory-based high-performance database in which all physical or real-world objects are represented by object instances of certain class types. The statuses of real-world objects are monitored by the WMI subsystem, and are stored directly as attributes in the appropriate objects in the RTOMS subsystem.

The RTOMS also allows definition of live links of RTOM objects to any other objects, either in the RTOMS *Object-manager* itself, or to any other object contained within a remote *Object-manager*. RTOMS also permits definition of derived objects, which is a logical or mathematical combination of any other objects' attributes.

The whole philosophy of RTOMS is that it is totally asynchronous and event-driven. For example, a real-world digital device changes state, WMI polls this state change, this change of state results in WMI sending a SOMI message to the digital device's object in RTOMS. The RTOMS objects' methods are triggered, which could result in further processing, linking 'output' attribute values to 'input' attribute values of other objects, and possibly sending SOMI messages to graphics objects contained within the MMI *Object-manager*. The change of attributes in the MMI *Object-manager* object could result in a defined colour change if that object was currently being displayed.

RTOMS Generic Requirements

For RTOMS, the following generic requirements were defined:

* It was necessary for RTOMS to store the entire real-world network image being monitored, viewed and controlled.
* Perform real-time processing associated with every object when it was triggered by any event or state change.
* For the required real-time response this real-time database had to provide high performance and throughput,
* To provide for dynamic linking of objects, so that a real-world stimulus of an object would trigger methods and processing of other linked and associated objects.
* To be logically contiguous, but possible to distribute this image database physically.
* Provide for regular image snapshots, so that the real-world image could be

recovered with minimal impact after a system crash.
* To provide a memory-based short-term history/profile archive which could be accessed in real-time.
* To support several types of enquiries.
* It had to model both the logical and physical object models.

RTOMS Design Considerations

The RTOMS subsystem is a complex and extensive portion of the operation-critical software. For this reason it would be impossible to review all the RTOMS issues. However, some of the design considerations are reviewed here since they relate to the object-oriented approach and the relevant decisions that were made.

It was observed that this was the most difficult subsystem to model, since it was the heart of the entire system, representing the entire outside world being monitored in terms of objects, both physical and logical representations. It also contained the object representations for the hierarchy of the real-world objects as well as derived objects.

In real-time applications, even with the traditional systems design methodology, the relational database has always proved unsatisfactory as the tool to store and manage the real-world image. The Relational Database Management System (RDBMS) cannot cope with the throughput, performance, real-time response requirements, and complexity of data types. The RDBMS cannot perform the complexity of operations, and does not support the concepts of dynamic links, linked lists and object attributes and object relationships that are constantly changing due to stimulus from events from outside events and alarms.

The usual approach to real-time databases has been customised memory-based hierarchical databases, or integration with packaged real-time databases. But this approach has resulted in designs and implementations that were very customised to the application - and therefore a very low degree of re-use. At best it has been observed that it is possible to re-use only the concepts and design of the RTOMS subsystem, and a small portion of actual code.

For the above reasons, use of an object-oriented real-time database made good sense. Several commercially available object-oriented databases were considered [1], such as ONTOS (Ontos Inc.), GEMSTONE (Servio Inc.), ORION (MCC) and Objectstore (Object Design Inc.).

However, it was decided to implement our own OODB for several reasons:

* Cost of proper evaluation of third party products.
* Learning curve to use these products.
* Fear about maturity and ruggedness of available products, particularly for real-time use.
* Reservations regarding the performance for real-time use.
* It would be difficult to optimise a third-party product for performance if this was required.
* Further reservations about security, integrity and recovery facilities provided - these are important criteria for a real-time database.

Thus, as our first venture into building object-oriented systems, it was decided that the risks were too high to use a commercially available OODB. It may well happen that once

the initial components of this project have been developed, evaluation and use of a commercially available ODBMS may be considered.

4.6.2 The RTOMS Building Block Structure

The preceding section outlined some of the generic *ObjectView* requirements of the RTOMS subsystem, and based on this the component structure of the RTOMS subsystem was designed. This is depicted in Figure 4.9 below.

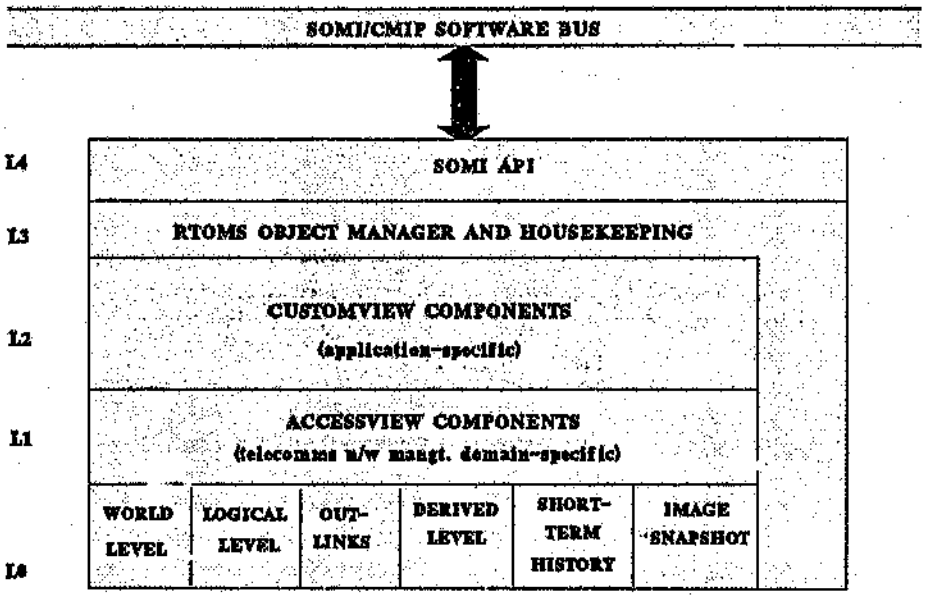


Figure 4.9: RTOMS Subsystem Building Block Structure

The component Levels L0 to L4 referred to in Figure 4.9 above are described below.

Level 0 (L0) These low-level components provide the *ObjectView* RTOMS components that are generic to real-time distributed applications. The WORLD ACCESS LEVEL component, also referred to as WAL, consists of the set of object classes that represent the physical real-world objects being monitored and controlled. The LOGICAL ACCESS LEVEL component likewise consists of the set of object classes that represent the logical representations of the real-world objects. The OUTLINKS component is responsible for providing the dynamic link mechanisms for linking object attributes to other local and remote *Object-manager* objects. The DERIVED LEVEL component consists of the set of object classes that

represent objects that are defined as derived, where a derived object is an object whose attributes are derived from a logical or mathematical combination of other objects' attributes. The SHORT-TERM HISTORY component consists of those object classes that handle the local high-performance memory-based short-term archive within the RTOMS subsystem. The IMAGE SNAPSHOT component consists of those object classes required for doing real-time database memory snapshots to disc, and executing rebuild recoveries.

Level 1 (L1) This level represents the *AccessView* Components that are built on top of the L0 components, and consist of components that provide abstracted and packaged functionality specific to the telecommunications network management domain. These components are covered in more detail in Chapter 5.

Level 2 (L2) This level is the *CustomView* component set that utilises the objects in the lower level components to provide the customised processing for the particular TNM application. An example is fault docket handling for a particular application. Further detail on the *CustomView* components is covered in Chapter 5.

Level 3 (L3) This is the generic RTOMS object manager with the associated housekeeping functions. The RTOMS *Object-manager* framework is very similar to the MMI and WMI object-managers, this being another advantage of the Software Bus concept coupled to the re-usability of the object-oriented approach.

Level 4 (L4) This is the SOMI Software Bus Application Programmer's Interface - the actual function calls providing access to the SOMI/CMIP Bus and services.

As with the MMI and WMI subsystems already described, these components were structured, contained and abstracted so that re-use, maintainability and testability are enhanced. As noted earlier in this section, it was observed that the RTOMS subsystem had the lowest level of re-use at the generic *ObjectView* layer, and it is my belief that this would hold true for any other application domains. The reason is that RTOMS really contains the representation of the logical and real-world models, and the behaviours associated with the particular application and application domain.

The RTOMS *AccessView* and *CustomView* components are described in Chapter 5, and further technical details on the modelling and implementation of the RTOMS subsystem are covered in Appendix D of this report.

4.7 CONFIGURATOR

The CONFIGURATOR subsystem is described in this chapter. In contrast to the RTOMS, it was observed that the CONFIGURATOR had some very clear and standalone generic *ObjectView* components. By standalone is meant that certain CONFIGURATOR components could be used as standalone re-usable program components.

4.7.1 The Generic Requirements of the CONFIGURATOR

Performance was the primary requirement. Three factors affect performance adversely in the CONFIGURATOR. The first is that the typical configuration database on a multi-networked system is usually contained on a single database server host computer, thus the bottle-neck. The second factor is that when developing software systems for end-users, there is often the constraint of having to adopt the database server technology used by the end-user corporation. These databases are usually Relational Databases which are notoriously slow in context of the real-time environment. The third factor is the large size of typical configuration databases.

The following basic generic requirements were defined for the CONFIGURATOR subsystem:

* It had to be flexible to allow update and access from any client object-managers on the network.
* To provide flexible integration into the Relational Database Management Systems (RDBMS), where the RDBMS database table design is often dictated by end-user corporate requirements. Often this design is not compatible with the real-time requirements, and even less so with the object-oriented paradigm.
* To allow for on-line real-time and batch configuration update.
* To provide for building of flat configuration files from the database tables for downloading to the relevant nodes where re-configuration of the RTOMS and WMI subsystems is required.
* Provide for partial (incremental) and full configuration building.
* It had to provide for the RDBMS to be taken off-line for periods for database administrator activities and maintenance. For reasonable off-line periods, the real-time system should not be adversely affected. That is, although the configurator service will obviously be unavailable, the real-time system should continue operation.

The above summarises the generic requirements of the CONFIGURATOR subsystem.

4.7.2 The CONFIGURATOR Design Issues

Before the component structure for the CONFIGURATOR subsystem could be designed, several fundamental design issues had to be reviewed:

- The first was why use a Relational Database for the CONFIGURATOR functions?
- The second issue which proved to be a big problem was that relational database technology and object-orientation are inherently incompatible.

Object-orientation implies encapsulation of object functionality with its data structures, whereas relational technology implies integrated data structures with orders of normalisation. This means that true object modelling does not necessarily have a one-to-one mapping between objects and database tables. These and other issues are discussed in further detail below.

Many technical real-time systems require configuration of some sort. Simply put, a configuration database is required which defines what the world outside looks like. This includes the mapping of the physical objects in the 'world' to 'logical' objects, with types,

characteristics, relationships and behaviours of the objects. The configuration also needs to define the behaviour of objects when certain events happen.

Listening to the object-oriented purists, we should have opted for a true Object Database Management System (ODBMS). However, this has been implemented to a degree for the RTOMS subsystem, which is the real-time database image of the real-world and derived objects. However a file-based configuration database is still required to provide backup to the RTOMS memory-based image. In this object-oriented approach, it was decided to use a configuration subsystem based on relational database technology.

The motivation to use relational database technology for the CONFIGURATOR was threefold:

* The first was pure economics and risks - developing or integrating a commercial OODB is expensive and risky.
* There is already heavy investment in relational databases in our organisation's typical end-users. The customers are not ready or prepared for a radical departure from this technology to object-oriented databases, especially where current relational databases are already integrated into their corporate environments.
* Powerful tools currently exist for reporting and database management. This includes the Industry Standard Query Language (SQL) and 4GL environments.

In the design of the building blocks, it was felt that there was place for both the OODB and the Relational Database. The RTOMS building block, described in the preceding section, is a real-time OODB, and it needs to be. This is because it is an active database - because almost all of the knowledge and code exists in the database itself.

The relational configurator database, however, is passive, and is intended to contain a file-based image of the real-world configuration, for RTOMS backup, system integrity and coordinated and synchronised re-configuration.

The ISO Product Data Exchange Specification (PDES) definition of data structures for design data, has already moved in the direction of object-oriented specifications for database. Object SQL, or OSQL is already being pursued by Hewlett Packard, Data General and others (Stone[52]). Most of these developments are based on adding object-oriented extensions to their existing SQL and other database products.

For similar reasons as deciding to implement our own RTOMS, for this project the team designed the concept of a Database *Object-Manager* (termed DBOM), which is an object-oriented layer *over* the relational database.

The component structure design of the CONFIGURATOR subsystem with the *Object-manager* layers is detailed in the following section.

4.7.3 The CONFIGURATOR Building Block Structure

With a conceptual understanding of the generic requirements of the CONFIGURATOR subsystem, and the related design issues reviewed, the subsystem building block structure was designed. This is depicted in Figure 4.10 below.

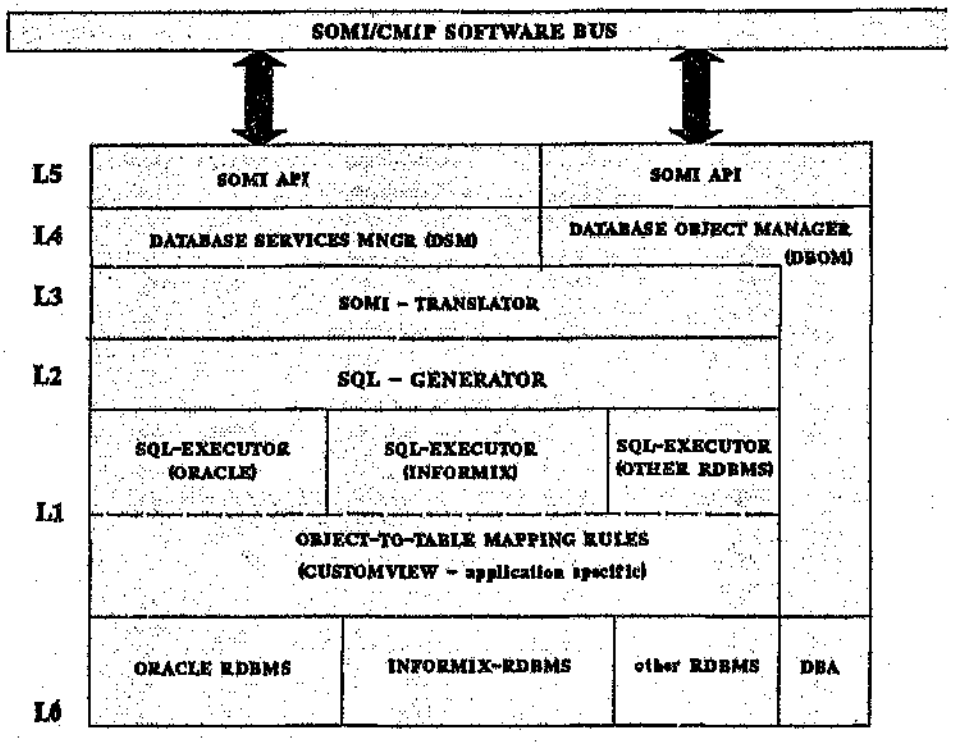


Figure 4.10: CONFIGURATOR Subsystem Building Block Structure

Figure 4.10 above depicts the structure of the main components of the Configurator subsystem. These levels are described in detail below.

Level 0 (L0) Level L0 represents the Relational Database Management System (RDBMS) products. In this project, use was made of the well established ORACLE and INFORMIX relational database products.

Level 1 (L1) This level consists of the SQL-Executor components. They consist of generic SQL-Executor components which provide the interfacing specific to the underlying database product. For example, the SQL-Executor (ORACLE) component provides a library of function calls which transform the upper layer standard SQL calls to the embedded Pro-C calls for ORACLE access.

Another feature of this level is the file-defined Object-to-Table mapping rules which is really a *CustomView* component, as it is very application specific. These *CustomView* components provide the mapping between the managed objects (MO) on the system and the tables in the RDBMS. As previously noted in this report, the object-oriented and relational

database technologies are inherently incompatible, and there is not necessarily a direct one-to-one match between object classes and database tables. Further detail on this *CustomView* mapping component is contained in Chapter 5.

In keeping with the re-usability of the object-oriented component approach, it was intended that any new RDBMS product may be used as the underlying storage mechanism by merely developing a new SQL-Executor component for interfacing to that database. The upper components and application would be unaffected by this.

Level 2 (L2) This level consists of a single component, the SQL-Generator, which converts the SOMI/CMIP primitive calls to SQL strings.

For example, the CMIP primitive: GET (class_name, object_name, ...)

would translate to the SQL string:
SELECT FROM <class> WHERE <object=object_name>

These strings are then passed to the lower SQL-Executor component which formulates the correct syntax for the embedded ProC interface calls to the RDBMS. This *ObjectView* component is generic to any application requiring database services.

Level 3 (L3) This is also a single component which unpacks the incoming SOMI service and translates it to the SOMI/CMIP primitives (GET, SET, CREATE, DELETE, etc). It is generic to any application requiring database services, and it was therefore designed as part of the *ObjectView* toolkit.

Level 4 (L4) Level 4 consists of the Database Services Manager (DSM) and the Database Object Manager (DBOM) components. These are *ObjectView* components as they are generic to any application requiring database services. The DSM receives all logon requests via the SOMI software bus from the requesting client application and schedules a dedicated DBOM for servicing subsequent database requests from the client application. Any new client application gets a dedicated DBOM. The reason for this DSM/DBOM arrangement is to be able to service requests from several client applications simultaneously.

Level 5 (L5) This is the SOMI Software Bus API - the actual function calls providing access to the SOMI/CMIP Bus, and SOMI/CMIP services.

4.7.4 Associated CONFIGURATOR Components

Apart from the above Configurator components, three additional stand-alone components were designed as part of the Configurator subsystem. These are the Application Configuration Manager (ACM), the Configuration Builder and the Bulk Loader. These components are reviewed in Figure 4.11 below.

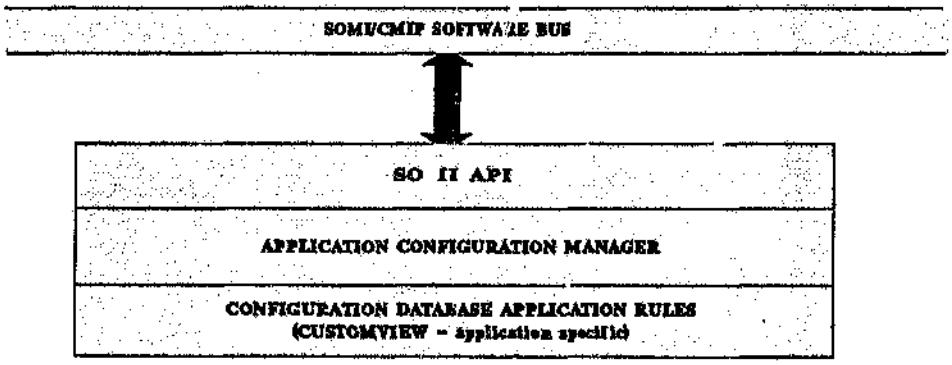


Figure 4.11: The Application Configuration Manager (ACM)

The Application Configuration Manager (ACM) subsystem shown in Figure 4.11 above consists of only 2 levels:

SOMI API

This is the SOMI Software Bus Application Programmer's Interface as described in previous sections.

Application Configuration Manager

This level consists of the generic *ObjectView* ACM component, and a *CustomView* application-specific database rules component.

Any access/update of the Configurator is done via the ACM to ensure the integrity of the database, and consequently of RTOMS, since RTOMS receives its image configuration from the Configurator.

The ACM itself was implemented as a state-machine engine, which operated on the file-based state-table Database Configuration Rules.

In this way, the ACM was kept generic, and the application-specific rules were kept as a separate file-based definition in the form of a state table.

The application configuration rules component is essentially a *CustomView* component. It is very application-specific, since it defines the rules for access and update of the Configurator database. These rules define the relationships, limits and hierarchy of all real-world and derived objects represented on the system.

Two other components associated with the CONFIGURATOR subsystem are the BUILDER and LOADER - these are depicted in Figure 4.12 below.

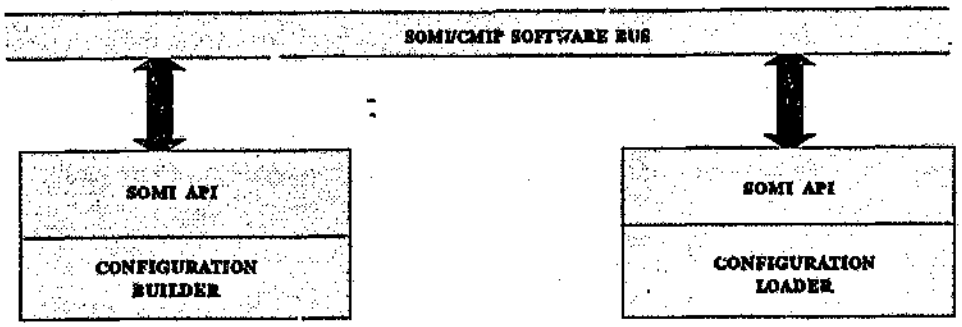


Figure 4.12: The Configuration Builder and Loader

The components shown in Figure 4.12 above are additional generic components associated with the Configurator.

The Configuration Builder

This component was designed and implemented as a generic *ObjectView* component which is activated when it is required to rebuild any RTOMS image on the network. A request is made for a partial or full rebuild from a user via the MMI. The MMI request is put on the SOMI software bus as a CMIP request to the destination CONFIG_BUILDER. CONFIG_BUILDER then accesses the RDBMS via the DSM/DBOM as a normal database client application, and builds SOMI-SYNTAX ASCII flat-files of configuration. After this is done, these local configuration files are then transferred to the host machine requiring an RTOMS re-configuration.

The Configuration Loader

The Configuration Loader was also designed and implemented as an *ObjectView* generic component. It reads the local configuration file (which is already in a SOMI-type syntax), and sends update requests to the local RTOMS via SOMI 'CREATE' and 'SET' messages. In this way the RTOMS memory-based image is configured from the configuration database.

4.8 HISTORIAN

The final subsystem is the HISTORIAN, which is described in this chapter.

As was the case with the CONFIGURATOR, the HISTORIAN also proved to have a high level of potential re-use, and all HISTORIAN components, except for the Object-to-Table mapping component, could be classified as generic *ObjectView* components. It is envisaged that the HISTORIAN components could be re-used in most real-time distributed systems requiring RDBMS historian and archiving functions.

The only non-generic component was the Object-to-Table Mapper component. It would not be re-usable in other real-time systems nor in other applications in the same telecommunications network management domain.

The design of the HISTORIAN and CONFIGURATOR was implemented in such a way so as to ensure re-usability between the two subsystems during development. In fact, the CONFIGURATOR was first developed, and thereafter it was relatively trivial to implement the HISTORIAN, since all components were essentially common except for the Level 4 components of CONFIGURATOR and HISTORIAN. These components are the HISTORIAN OBJECT-MANAGER (HOM), and the DATABASE OBJECT MANAGER (DBOM). The DBOM for the CONFIGURATOR was designed as a generic database server object-manager/agent, but the HOM was designed as an extended DBOM with functionality added to cater for specific historian requirements.

The HISTORIAN components are described in the following sections of this chapter.

4.8.1 The Generic Requirements of the HISTORIAN

The following generic requirements were defined for the HISTORIAN subsystem:

* As with the CONFIGURATOR, performance was the primary requirement. It is necessary for several client object-managers or applications (usually RTOMS object-managers) to be able to archive or log data to the HISTORIAN server host in real-time. It was necessary for the HISTORIAN to handle rates of up to several dozen 'object-records' per second.
* The HISTORIAN should be able to store and manage large volumes of data - for the TNM application developed it had to cater for up to 3Gbytes of data.
* It had to include a store and forward mechanism, meaning that data could not be streamed directly into the Historian database server. A buffering mechanism between the sending object-manager and the HISTORIAN had to be provided. This would cater for situations where the database itself was down or the network connection between the sending object-manager and HISTORIAN database server were down.

With the basic HISTORIAN requirements defined, the component structure of the HISTORIAN subsystem was designed.

4.8.2 The HISTORIAN Building Block Structure

Having defined the generic requirements of the HISTORIAN subsystem, the HISTORIAN Component structure was designed. This is depicted in Figure 4.13 below.

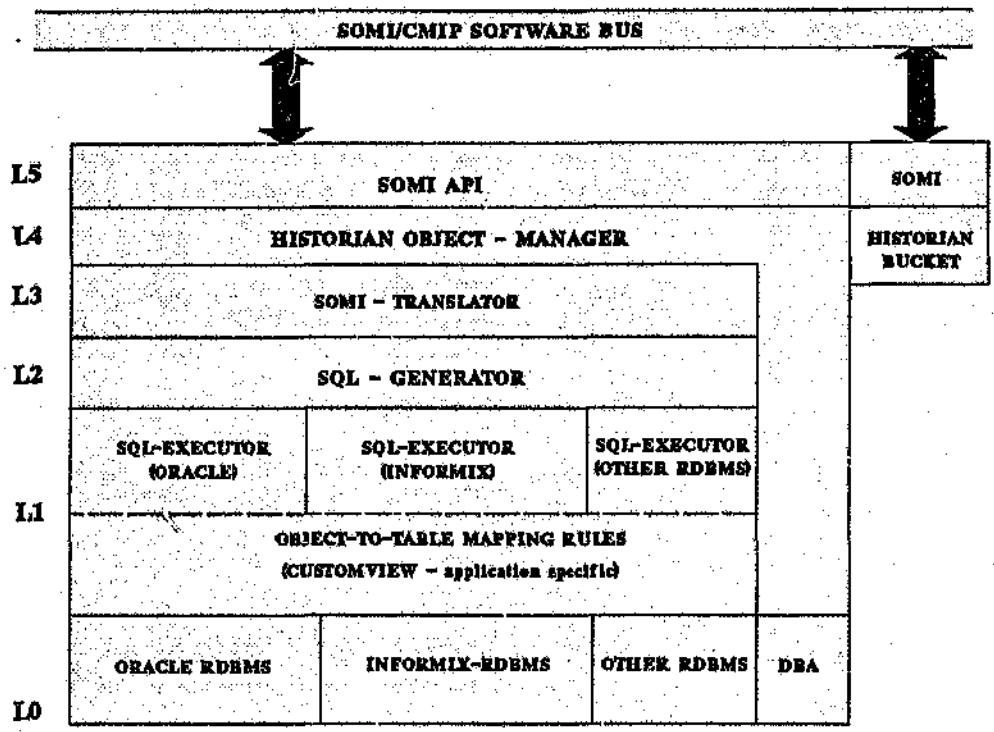


Figure 4.13: HISTORIAN Subsystem Building Block Structure

Figure 4.13 above depicts the structure of the HISTORIAN subsystem. These levels are described below in detail below.

Level 0 (L0) Level L0 represents the Relational Database Management System (RDBMS) products. In this project, use was made of the well established ORACLE and INFORMIX relational database products.

Level 1,2,3 These are the generic SQL-Executor, Object-to-Table mapping, SQL-Generator and SOMI-Translator components and are identical to that of the CONFIGURATOR subsystem. Refer to the descriptions of these components under the CONFIGURATOR in Chapters 4 and 5.

Level 4 (L4) Level 4 consists of two *ObjectView* generic Historian components: the Historian Object Manager (HOM) component and the Historian Bucket component. The HOM behaves in almost the same way as the CONFIGURATOR's DBOM object-manager component. That is, it receives historian data messages from object-manager clients from around the network via the SOMI software bus. It then transforms these into SQL statements that are then executed in the RDBMS.

A HISTORIAN BUCKET application resides with each historian client object-manager (usually RTOMS's) and all historian bound messages are routed to this Bucket. The bucket stores these archive messages and handles its own interaction with the HOM object-manager on the Historian Server host. When the HOM is ready for the next database 'insert', it requests the next historian record from each of the running BUCKET applications in turn. Once received, the HOM then stores it in the database.

The reason for the store and forward mechanism of HISTORIAN is to prevent performance critical object-managers such as RTOMS from waiting for returns from relatively slow database accesses. The BUCKET-HOM interaction provides a secure buffering and handshaking mechanism, and off-loads from the RTOMS the problem of ensuring successful archive storage to the BUCKET application. The BUCKET also provides for extensive buffering for cases where the database server host is temporarily unavailable due to maintenance.

Level 5 (L5) The SOMI Software Bus API, as previously described.

4.9 UTILITIES

Several utilities had to be implemented as part of the generic components required for building real-time distributed systems. These utilities are seen as *ObjectView* components, as they are generic to any application domain.

The more important utilities are briefly reviewed in this section, but no additional details and designs are included in this report for the sake of brevity.

The extreme benefits of the object-oriented approach coupled to the SOMI software bus became apparent with the addition of further utilities and functions. Most utilities were *ObjectView* Applications which communicated to any other object-managers or applications via the CMIS services provided by SOMI. This resulted in:

* Generality of utilities, allowing use of a utility by other object-managers or applications anywhere on the network.
* Transparency - as with object-managers and applications, utilities are just applications and may be run transparently on any node on the network.
* Ease of re-use and development - writing a new utility involves re-using the standard Application template and then writing the utility code within this template.

Error-log and System-log

These subsystems are required in any real-time system, and are used to store error logs and system logs generated by the object-managers and application processes themselves. Abnormal software errors encountered by processes on the network log 'error logs' to the Error-log utility on the network, and any process wanting to log a change of operating state or condition, logs a 'system log' to the System-log utility. These logs also store, index and manage the log files and allow easy and efficient retrieval of logs from any requester on the SOMI bus.

The error-log utility receives error logs from any object-manager or application process on the network via the SOMI bus, and typically includes time and date of the error, object-manager/application name, node, error code and description.

The System-log is identical to the Error-log, except that it stores information on events within the object-manager and applications, such as start-up time, status and condition of processes. The information in the System-log also include the date and time of log, object-manager/application name, node and log message string.

Figure 4.14 below depicts the SOMI bus with object-manager processes and the Error-log and System-log applications which receive, store, and allow retrieval of the logs.

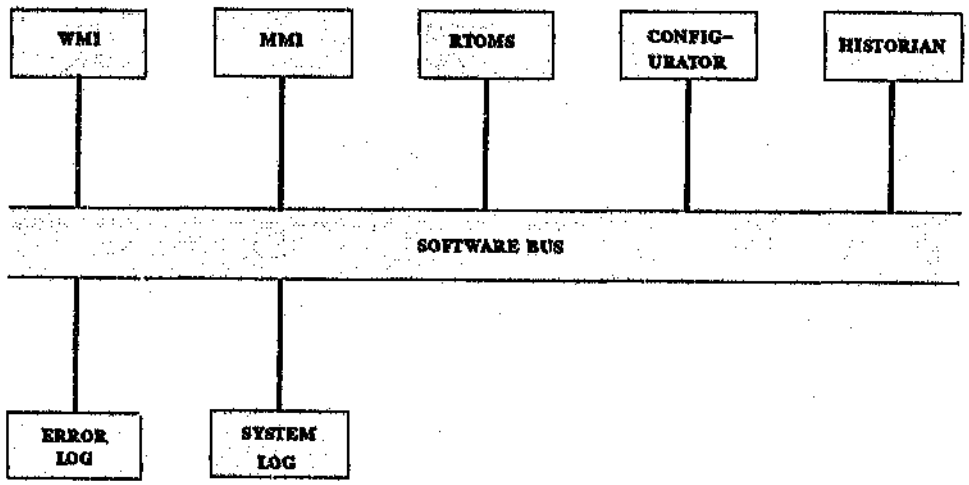


Figure 4.14: The Error-log and System-log Utilities

Generic Watchdog

The generic watchdog is also is an Application attached to the SOMI bus and runs on one of the nodes on the network. According to system manager defined parameters, it regularly sends SOMI 'GET' requests to the defined object-manager processes on nodes on the network to enquire operational status. Responses are returned to the Watchdog utility and necessary actions and logs are sent out by the watchdog when error conditions or non-responses are detected.

For this watchdog utility, generic watchdog mechanisms had to be set up in the

object-manager processes. Firstly, all objects being managed (MOs), were designed to contain status attributes, and methods to access these attributes. Secondly, each generic object-manager was designed to contain a special class named 'STATUS' which itself inquires and stores the status of the required MOs in that process, and derives the operational status of that object-manager.

Thus the generic watchdog utility actually addresses the 'GET' request to the STATUS object of each object-manager and so derives the status of the entire system.

SOMI-Send Utility

This utility was developed to aid in debugging, testing and integration of the components of *ObjectView* and *AccessView* and was also very useful in integrating and testing the completed TNM application that was developed. It was written as an application which attaches to the SOMI bus. Via interactive user input or input from a script file, it sends SOMI messages (GET, SET, CREATE etc) to the addressed object-manager on the network. Responses are returned to the invoking SOMI-Send utility, which can then be analysed and browsed.

4.10 CHAPTER SUMMARY

This chapter has primarily focused on the introduction of the generic architecture, and the design of the core generic components (*ObjectView*). Particular attention was also given to the use of the software bus and highway.

It is strongly believed that the designed structure of the generic components has confirmed the suitability of the object-oriented approach for software re-usability. Furthermore a 3-layered component structure was introduced and motivated. It was demonstrated in this chapter that this, together with the SOMI software bus, further contributed to the success of the object-oriented component-centred approach.

It has been demonstrated in this chapter how the major components were further sub-divided into smaller well defined sub-components or building blocks, which incorporate the special requirements of distributed real-time systems. However, for the sake of clarity and relevance, detailed examples of the real-time data modelling and implementation of these components have been omitted from this chapter, but an example is contained in Appendix D of this report.

5. BUILDING THE DOMAIN AND APPLICATION COMPONENTS

This chapter further describes and demonstrates the object-oriented component approach to building the TNM applications. Chapter 4 detailed the design and implementation of the first layer (*ObjectView*) of the 3-layered building block design philosophy. In this chapter, the design and implementation of the second and third layers (*AccessView* and *CustomView*) are described, where the *AccessView* components are those that are TNM domain-generic, and the *CustomView* components are those that are TNM application-specific.

Again, specific emphasis was made on designing components for re-use, maintainability and testability, without falling into the trap of over-generalising the functionality of components.

Only that material of the *AccessView* and *CustomView* components that is relevant to the report topic is covered in this chapter. However, Appendix D does contain examples of the modelling and implementation of some of these components.

5.1 REVIEW OF THE *ACCESSVIEW* APPLICATION DOMAIN

The *AccessView* domain referred to in this report is contained to a specific portion of the Telecommunications Network Management domain. This domain is extensive and explanation in detail is beyond the scope of this report. However to better illustrate the structure, design and function of the *AccessView* object-oriented components, this section contains a more detailed overview of the real-world domain of telecommunications networks applicable to this study. It should be regarded as a follow-on to the description of the TNM domain introduced in Chapter 1.

5.1.1 The Physical Model

In context of this study, there are three relevant areas of telecommunications networks, namely line systems, transmission systems and switching. These have been previously introduced in Chapter 1 - refer to Figure 1.1 for a depiction of these three relevant areas of telecommunication networks in relation to a telecommunications network management system. Descriptions of these three areas are restated below.

Line Systems - Carries telecommunications traffic via microwave, fibre-optic and other line systems. The electronic equipment associated with the line systems has outputs that indicate the operational and error status of that equipment. These indications are monitored and stored by the remote data acquisition outstations (eg. OCTAVE 2000, RICE-80) situated in the carrier rooms, and are periodically polled by the regional WMI hosts.

Transmission - This consists of 2, 8, 34 and 140 Mbit stream multiplexer and demultiplexer equipment, which also has outputs indicating the operational and failure status of that

equipment as well as bit-error indications. These are also monitored by remote outstations which are periodically polled by the regional WMI hosts. The transmission and line subsystems are merely bearers of traffic they are carrying.

Switching - These are the end users of the transmission and line systems. Data acquisition outstations for monitoring the relevant switching equipment are situated in the switching exchanges. These outstations periodically poll the exchange equipment and monitor the faults and status of the switching equipment.

The Telecommunications Networks Physical Model - Figure 5.1 below depicts the physical model of Figure 1.1 in greater detail. For the sake of brevity, the switching areas of the telecommunications networks have been omitted.

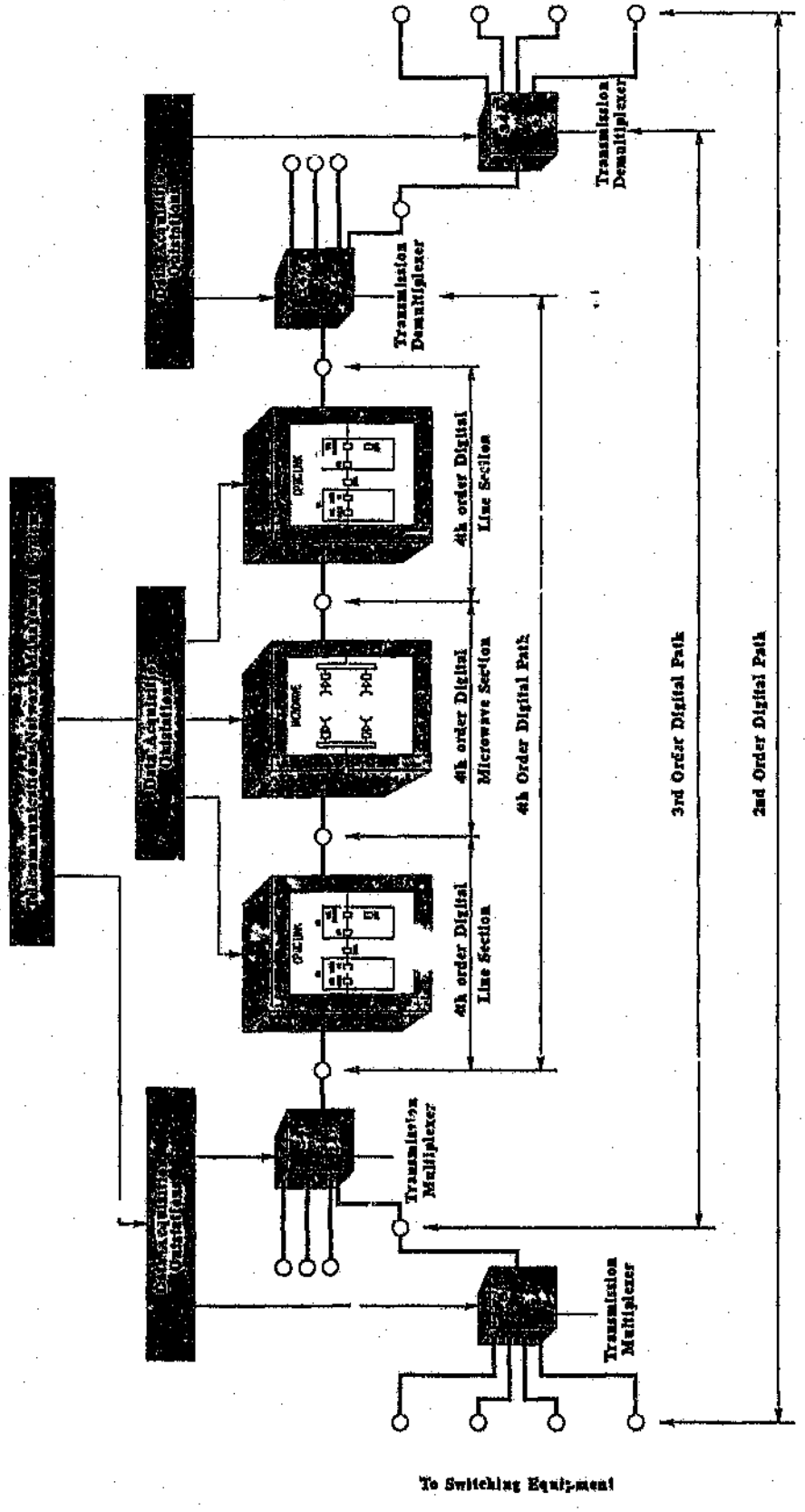


Figure 5.1: Telecommunications Networks Physical Model Example

The Physical Equipment Hierarchy - Figure 5.1 depicts the physical *model* of telecommunications networks, the *hierarchy* of which is depicted in the physical hierarchy diagram depicted in Figure 5.2 below:

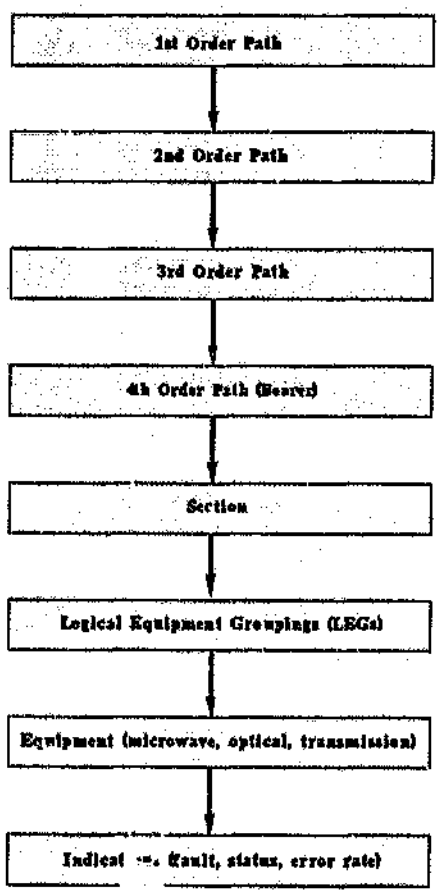


Figure 5.2: Telecommunications Networks Physical Hierarchy

The nature of the physical telecommunications network model is that the configuration of equipment within the above hierarchy is dynamic. For example, a fourth order Bearer may consist of certain LEG-sets at one time, but later could be configured to consist of different physical LEG-sets. This dynamic configuration is applicable to all elements in the hierarchy.

5.1.2 The Object Model

Figure 5.3 below depicts a Booch representation of the object model of the telecommunications networks domain. This simplified object model is a representation of real-world physical model and relationships - and relates directly to the physical model depicted in Figure 1.1 of Chapter 1, or Figure 5.1.

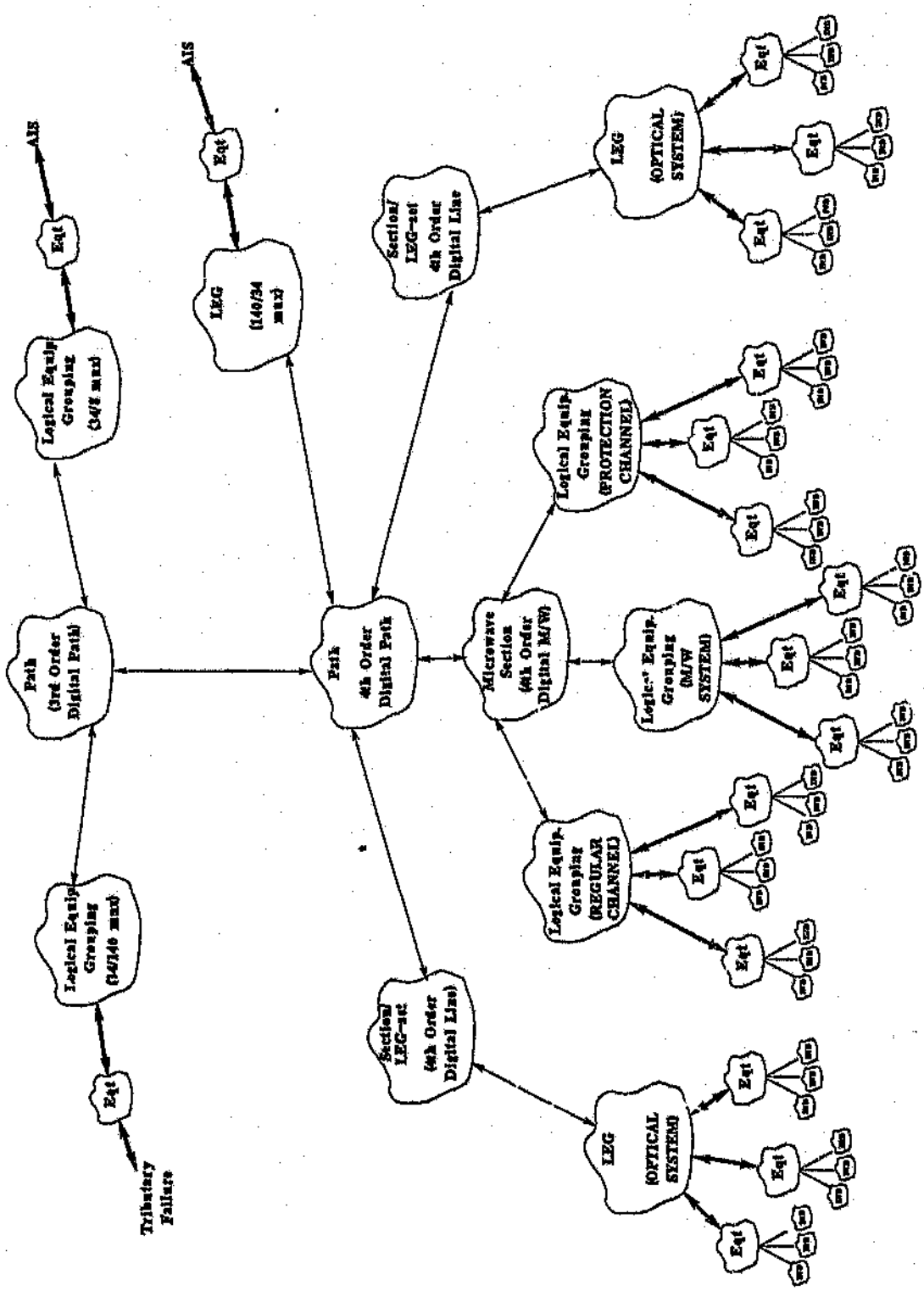


Figure 5.3: Telecommunications Networks Object Model Example

5.1.3 Generic TNM Functions

From the domain overview covered in the preceding sections, it is easy to appreciate that such extensive transmission equipment requires integrated and effective management. With successful Telecommunications Network Management (TNM), a high level and grade of service is ensured to all users of telecommunications, from private telephone subscribers, to high-speed corporate/banking customers.

So that the reader may better appreciate the structuring and design of the *AccessView* components described in later sections of this chapter, the generic functions of TNM systems are summarised below:

* Monitor equipment failures on microwave and optical line systems, transmission and switching equipment (referred to as Fault Management)
* Monitor statuses and maintain configuration on line, transmission and switching equipment (to be classified as Equipment Management).
* Monitor the grade (bit-error-rate) on the bearers or channels (also Equipment Management).
* Dynamically add, delete and reconfigure the telecommunications network configuration on the TNM system (Configuration Management).
* Monitor traffic (usage and utilisation) being carried by different bearers and links (referred to as Traffic Management).

The above concludes the overview of the telecommunications network domain, including what is meant by lines, transmission and switching, and the physical relationships between the different types of equipment. This overview has been included so that the reader can better appreciate the following sections which describe in detail the designed structure of the *AccessView* and *CustomView* components.

5.2 THE MAN-MACHINE INTERFACE

In this section the network management domain-specific components, that is *AccessView*, of the Man-machine Interface are described. This layer was designed to encapsulate the functionality that would be required for typical telecommunications network management applications, specifically fault, equipment and traffic management.

The extended software specific to the particular application (referred to as *CustomView*), is also described. For the sake of clarity, further description and technical details of the design, modelling and implementation of the MMI's *AccessView* and *CustomView* components has been omitted from this project report.

Below, Figure 5.4 shows the same MMI component structure given in the *ObjectView* description in Chapter 4, but with a breakdown of the *AccessView* and *CustomView* components.

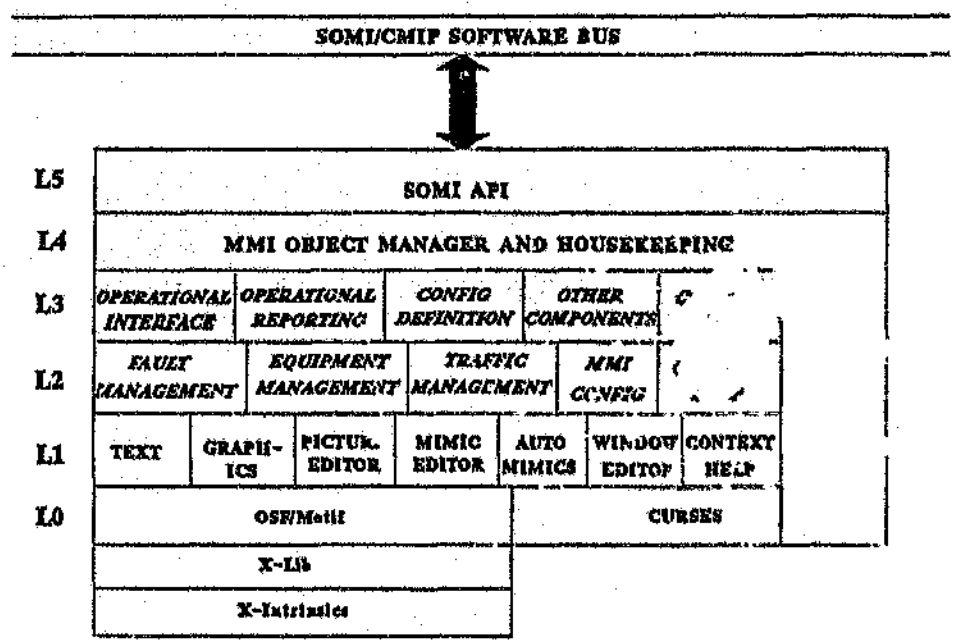


Figure 5.4: MMI AccessView/CustomView Components

5.2.1 The AccessView Components

These are the domain-generic components. This is the level where the semantic level of components associated with the telecommunications network management domain are provided.

Level 2 (L2) in the Figure above shows these *AccessView* components. In reality there are many more detailed components at this level, but for the sake of relevance to the subject of this study, the components at this level have been summarised into four basic components as described below.

Fault Management: This component contains the set of abstracted objects with the encapsulated functional behaviour for the real-time display and user-manipulation of faults and fault dockets. It can really be seen as an extension to the Text and Graphics components of the MMI *ObjectView* layer, and includes the functionality for retrieving all

summary and detailed fault and docket information from the RTOMS via SOMI message requests, and manipulation and display of this information on the screen. (Refer to the RTOMS section in this chapter for the definition of faults and dockets).

Refer to Appendix F for examples of class definitions of objects developed for fault display management in the MMI subsystem. Refer specifically to:

* Include file *DktSummary.h* which provides for summary displays of fault dockets,
* Include file *MImDisplay.h* which provides for general display functions peculiar to telecommunications network management.

Equipment Management: This component is also an extension to *ObjectView*'s Text and Graphics components object classes, and provides the functionality for the real-time enquiry and display of status of real-world objects as stored in RTOMS. The functionality includes the MMI-RTOMS interaction for retrieving the real-world object statuses via SOMI requests, and real-time display on the screen.

Traffic Management: The traffic management component contains the objects and functionality for displaying traffic information relating to the links in the telecommunications network. This traffic information includes traffic volume, traffic users, and link and bearer quality, performance and utilisation.

Refer to Appendix F for a class definition example of a traffic performance object, defined in include file *LinePerfDisp.h*.

All three of the above *AccessView* components include functionality for reporting of archives from either the long-term archive in the HISTORIAN subsystem, or the short-term archive contained within the RTOMS subsystem.

MMI-Configurator: The configurator component includes the MMI functionality for entering and updating the telecommunications network configuration in the Configurator subsystem. For example, the configuring of digital and analogue indications to equipment, equipment to logical equipment groupings (LEGs), LEGs to sections, and sections to paths.

5.2.2 The CustomView Software Layer

Level 3 (L3) in Figure 5.4 above shows the *CustomView* application-specific components. These components are not really generic and are customised for a particular TNM application, since they embody semantics very specific to the particular application.

Again as with the MMI's *AccessView* components, there are many more detailed components at this level, but for the sake of relevance only three basic *CustomView* components are described:

Operational Interface: This component provides the look and feel of the graphical (and character) operational user interfaces required for fault, equipment and traffic management. It defines the windows arrangement and representation and presentation of telecommunications network objects on the screen. It includes objects that display the current network configuration, inter-relationships and equipment hierarchy.

Operational Reporting: Operational reporting provides for the display and management of the archives, the short-term archive from the RTOMS subsystem, and long-term archive from the Historian subsystem.

Graphics-configurator Definition: This component provides the application-specific definition of what the graphical and character screens would look like for the configuration.

The component philosophy of these *AccessView* and *CustomView* components is the same as for the *ObjectView* components described in Chapter 4 - abstracted object component libraries that are "hooked" into the generic *Object-manager* process.

5.3 THE WORLD-MACHINE INTERFACE

In this section, the telecommunications network management specific components of the WMI are described. Referring to the WMI structure described in the previous chapter, these are the *AccessView* components represented by Level 1 (L1) components in the subsystem structure design. For the initial phase of the project, three components were designed and implemented. This was required for driving three specific types of telecommunications data acquisition outstation equipment, the OCTAVE 200, OCTAVE 2000 and the RICE-80. (Further details on the outstation equipment is contained in the description of the *AccessView* components below).

The next higher level 2 (L2), the WMI *CustomView* components, are also described. These components represent the customised portions of the software specific to the particular application that was developed. Again, for the sake of relevance, further technical details on the design, modelling and implementation of the WMI's *AccessView* and *CustomView* components have been omitted.

Below, Figure 5.5 depicts the WMI component structure with the breakdown of the *AccessView* and *CustomView* components.

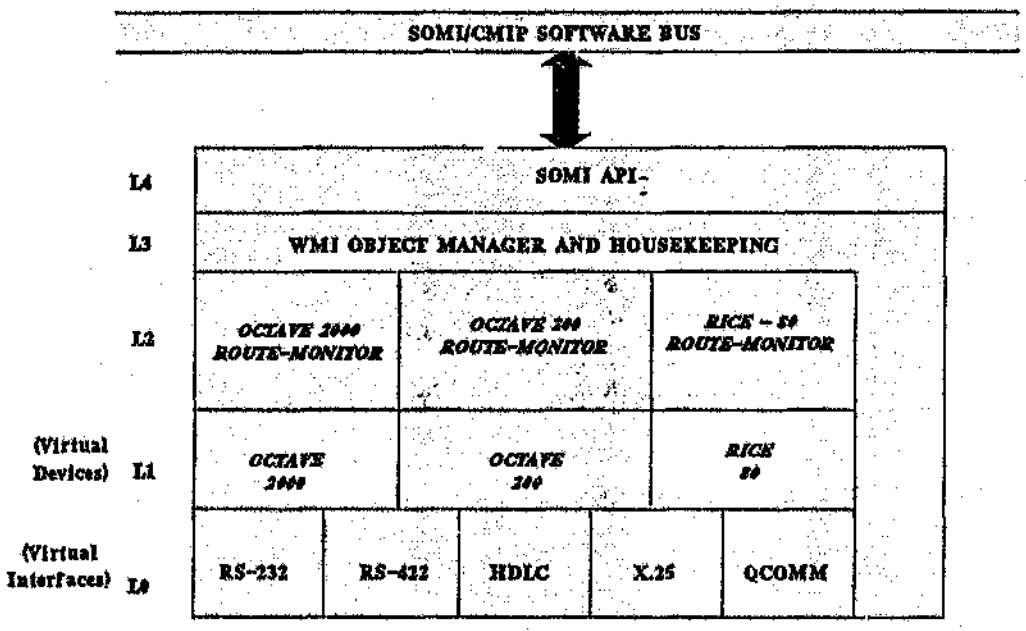


Figure 5.5: WMI AccessView/CustomView Components

5.3.1 The AccessView Components

This level (L1) is the *AccessView* abstraction of the WMI subsystem, and the Virtual Devices shown in the Figure above represent a few of the relevant devices used as outstations in telecommunications network management systems. For this project, only three devices were implemented; the OCTAVE 2000, the OCTAVE 200 and the RICE-80 outstations.

The Physical Outstation Devices

To better understand the structure and functions of the WMI interface components, a brief review is given of the outstation hardware devices used in this project.

These outstations monitor digital and analogue indications from transmission, optic, microwave and multiplexer and bearer equipment. They are usually physically situated in transmission carrier rooms and exchanges around the country. The indications monitored relate the operational state and condition of the transmission equipment, for example communications break, power fail, and degraded bit-error rates in transmission.

More specifically, the OCTAVE-2000 is a proprietary intelligent monitoring and control outstation used for monitoring digital and analogue indications from Transmission and Microwave equipment, as well as bit-error rates (BER) associated with transmission systems. These digital and analogue indications are used to indicate transmission and line faults.

The OCTAVE-200 outstation device is an older version of the OCTAVE-2000, and is a

non-intelligent monitoring outstation that monitors digital and analogue indications.

Lastly, the RICE-80 outstation device is another type of semi-intelligent proprietary device for monitoring transmission and microwave equipment in telecommunications networks.

The WMI Virtual Devices

On a conceptual level, the OCTAVE-2000, OCTAVE-200 and RICE-80 Virtual Device components behave and perform in the same way. They poll and receive data from the relevant remote outstation devices, as well as sending enquiries and controls and configuration messages. The major difference between these Virtual Device components is the protocol syntax and semantics which are obviously very specific to the relevant hardware outstation devices.

5.3.2 The CustomView Software Layer

Level L2 of Figure 5.5 represents the *CustomView* abstraction or customised application-specific components. Each is referred to as a Route-Monitor, since each is responsible for talking to one route, which is physically a single multi-dropped or point-to-point line to which addressable outstations are connected to at the remote locations.

Each of these components contain the following basic functionality:

* Poll of outstation on multi-dropped line using intelligent sequencing algorithms.
* Enquiry of outstation status.
* Control of configuration downloads to outstation devices.
* Control of embedded software downloads to outstation devices.

5.4 THE REAL-TIME OBJECT MANAGEMENT SYSTEM

In this section, the telecommunications network management specific components of the RTOMS are described. The RTOMS subsystem structure given in Chapter 4 shows level 1 (L1) to represent the *AccessView* components, and level 2 (L2) to represent the *CustomView* components. In this section, the level 1 and 2 components are described, addressing the fault, equipment and traffic management.

Further technical details and examples of the design, modelling and implementation of the RTOMS *AccessView* and *CustomView* components are contained in Appendix D.

Below, Figure 5.6 shows the RTOMS component structure given in Chapter 4, but with a breakdown of the *AccessView* and *CustomView* components of RTOMS.

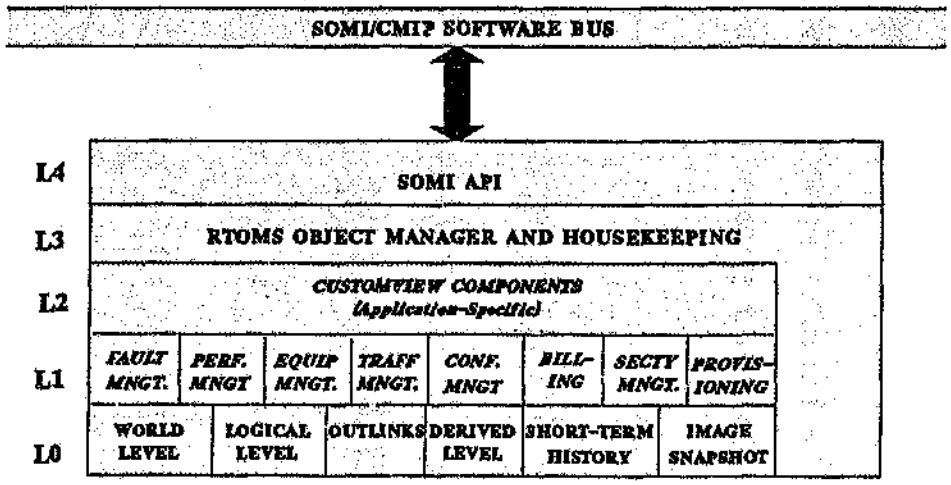


Figure 5.6: RTOMS AccessView/CustomView Components

5.4.1 The AccessView Components

Level 1 (L1) in the Figure above depicts the design of the RTOMS *AccessView* components, and these are briefly described here.

Fault Management: This component provides the set of defined objects for handling real-world equipment events and faults as monitored by the WMI. All faults are tracked and are associated with a 'fault docket'. The fault docket tracks the status and all information relating to the fault and its associated faults. The tracked information includes time of fault, type and location, action taken for fault, time of fault clear, reason for fault clear, and other faults that may have been related to this fault.

Refer to Appendix F for the class definition of the fault docket object of the RTOMS subsystem. The include file is named *Docket.h*.

Equipment Management: The equipment management component is the extension to the World Level (WAL) *ObjectView* component of RTOMS that provides the functionality to store, derive and report the status of all real-world equipment, where all real-world equipment indications are represented by objects in the World-Level component.

Traffic Management: The traffic management component contains the objects for monitoring, measuring and reporting the quality and level of telecommunications traffic flowing through the telecommunications transmission and microwave links and paths. It also includes the functionality for reporting link utilisation. All three the Fault, Equipment

and Traffic management components include functionality for storage of required information to the long-term HISTORIAN archive or the RTOMS memory-based short-term archive.

Performance Management: This component is an *AccessView* extension of the objects in the Logical-Level component of RTOMS. It provides extensions to the objects representing the paths in the logical-level's 'indication-equipment-section-path' hierarchy. These extensions provide for performance monitoring and measurement of all the links on the system.

Other *AccessView* Components: Configuration, Security, Billing, Security Management and Provisioning are other *AccessView* components that are not described any further in this report. They are relevant to the telecommunications network management domain, and are currently seen as components that could be developed in the future.

5.4.2 The CustomView Software Layer

Level 2 (L2) in Figure 5.6 above depicts the *CustomView* application-specific components of RTOMS. During design of these components, it was observed that this layer was very thin - since the *AccessView* components themselves contained much of the application-domain and application-specific knowledge.

It is intended to deliver similar systems to two other applications in the same domain over the next two years. Analysis of their requirements has proved that such a low level of RTOMS re-customisation is required, that the differences have already been incorporated generically into the *AccessView* components objects.

5.5 THE CONFIGURATOR

In this section the non-*ObjectView* components are described. Referring to the Configurator Component Building Block Structure in Figure 4.10 of Chapter 4, it is observed that no *AccessView* components exist in this subsystem, and only two *CustomView* components exist. Of all the subsystems, the Configurator proved to be the most generic subsystem, being entirely generic to other systems requiring configuration services except for two *CustomView* components which are described below.

5.5.1 The CustomView Software Layer

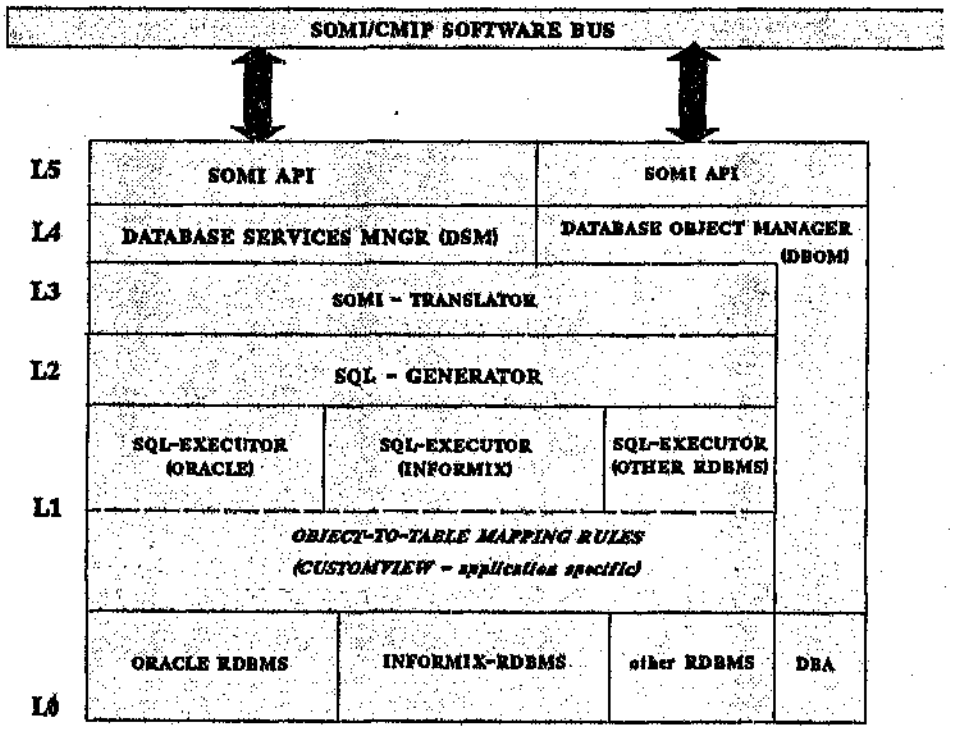


Figure 5.7: Configurator Customview Components

The two *CustomView* components of the Configurator were the Object-to-Table Mapping Rules which is part of level L1 of the Configurator building block structure diagram (Figure 5.7 above), and the Application Configuration Rules (shown in Figure 5.8 below), which is associated with the ACM Configurator component.

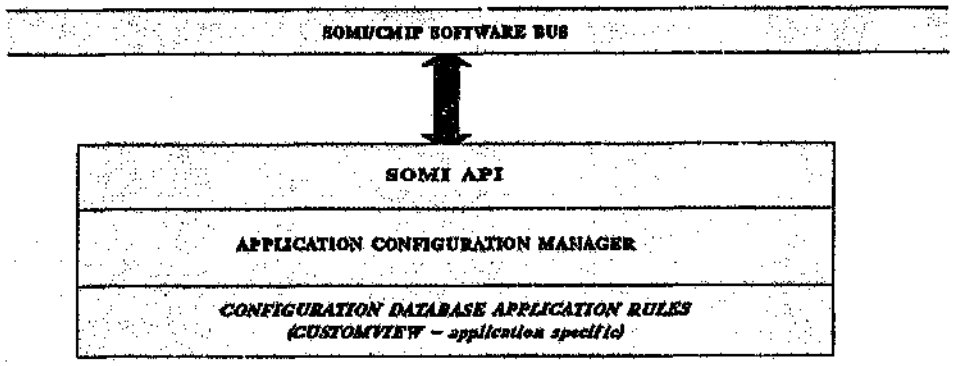


Figure 5.8: The Application Configuration Manager

The Object-to-Table Mapping Rules Component

This component is very application specific and defines the object-to-table mapping rules in a user-definable ASCII definition file. This definition file provides the definition for the mapping between the managed objects (MO) on the system and the tables in the RDBMS. As previously described, the object-orientation and relational database technologies are inherently incompatible, and there is not necessarily a direct one-to-one match between object classes and database tables.

The object-to-table mapping rules could also be viewed as objects, where each class (eg. indications, equipment, logical equipment groups) that needs to be mapped to database tables, could encapsulate its own mapping rules. However in practise it was not implemented as C++ objects, since it is a requirement to be able to alter the mapping rules without having to edit and recompile C++ programs.

For this reason the object-to-table mapping rules were defined in ASCII files via the UNIX file editor. An example of the syntax of this mapping definition file is given in Appendix E of this report.

The Configuration Database Application Rules Component

The Application Rules component is a ASCII file-based set of application-specific rules for accessing and updating the Configuration database. It is closely associated with the ACM which is the state-machine engine which reads and operates on the rules and definitions contained in the Configuration Rules. These rules define the relationships, limits and hierarchy of all real-world and derived objects represented on the system.

5.6 THE HISTORIAN

As with the Configurator subsystem, no Historian *AccessView* components were designed, and likewise the only *CustomView* component was the Object-to-Table mapping rules. As already noted in the previous chapter dealing with the Configurator and Historian, the generic *ObjectView* components of these two subsystems are identical. This is readily observed by studying the Component Structure Diagrams of these two subsystems in Chapter 4.

5.6.1 The CustomView Software Layer

As with the Configurator, the only *CustomView* component of the Historian subsystem is the file-defined Object-to-Table mapping rules which is application specific. This component provides the mapping between the managed objects (MO) on the system and the tables in the RDBMS. An example of the syntax and operation of the Object-to-table definition file is given in Appendix E of this report.

5.7 CHAPTER SUMMARY

This Chapter has detailed the structure and design of the *AccessView* and *CustomView* components which were designed and developed for an application in the telecommunications network management domain.

It must be noted that the entire scope and detailed design and implementation of the application has not been covered in this report. Since this study concerns an object-oriented approach to building real-time systems, discretion was used to include only that material which would demonstrate the process and results of this approach, and thus the focus is not on the application itself, nor the detailed implementation details.

Chapter 5 focused on the object-oriented component approach for the domain-generic and application-specific layers. The design has demonstrated that it is possible to optimise component or building block structuring for optimal re-usability, maintainability and testability. This chapter has also highlighted the effort that was expended in the design and structure of the components required for the real-time systems.

Furthermore in this chapter the effectiveness and practicality of the 3-layered component structuring has been demonstrated. These 3 layers were the real-time system generic components, domain-generic components and then application-specific components. It proved to be very effective because re-use was optimised, and the 3-layered structuring also provided sensible and practical abstractions.

6. ANALYSIS OF APPROACH

At this point the object-oriented component-based approach to building real-time systems has been motivated and described. The design of the building blocks or components, and the integration of the components in building a telecommunication network management application has also been described. In this final chapter, the results of this approach are analysed and observations of relevant issues are made.

In the final analyses, some of the motivations put forward at the beginning of this report are corroborated and others are challenged. Likewise, observations made by noted authors on the related topics are also corroborated or challenged.

In terms of the focus of this study, namely an Object-oriented Component-based Approach to Building Real-time Systems, I have intended to put forward three arguments:

* That the object-oriented paradigm proved to be a successful approach for building real-time systems,
* That the 3-layered structured component approach coupled with the CMIP-conformant software bus further enhanced the success of the object-oriented approach,
* That several serious technical and organisational issues need to be noted when considering this approach.

In this final chapter, it is intended to use the results and observations of the study as a basis for the motivation, proving and endorsement of the above arguments.

6.1 FIT WITH CUSTOMER REQUIREMENT

One of the criticisms levelled at the traditional procedural approach to building custom software systems is the fit with the customer requirement. These criticisms were reviewed in an earlier chapter of this report. Of particular note was the accuracy with the original requirements were interpreted, and the fact that requirements do change. This change takes place during the entire life-cycle of the project, from specification to well after delivery.

The fact that the traditional top-down approach takes no account of evolutionary changes has been observed and noted by several authors including Lehman[40], and is also confirmed by personal project experience. In fact, the later in the project life-cycle changes to requirements occur, the more severe the impact to the design, integrity, and re-development effort.

From the observations of this study, the object-oriented approach proved to be far more amenable to changing requirements. This was largely attributable to two factors:

* The object-oriented life-cycle model (refer to the Object-oriented life-cycle, Chapter 3) is a tight iterative coupling between the modelling, design, code prototyping and testing phases.

* The object-oriented paradigm is based on abstraction and encapsulation, with message interfacing between objects as opposed to data-structure interfaces. Changes made to any objects behaviour had minimal effect on other objects.

Because of the properties of object-orientation and the iterative life-cycle model, changes could be locally modelled, prototyped and tested very effectively with very little impact on the associated objects and software subsystems.

During the design, development and testing phases of this TRIM project's lifecycle, several new requirements were presented. In very few cases was it necessary to call together the entire design team to discuss the implications of a functional software change. In each case, the new requirement was directed to the relevant person/s on the team.

In summary, the study clearly observed that the object-oriented approach was far more suited to changes than the traditional procedural approach. This has resulted in a delivered system with a close fit to the end-user requirement.

In fact, the object-oriented approach proved so amenable to changes, that it brought with it a new set of problems. Because changes were easily implemented, the formal software change request/proposal procedures were sometimes bypassed, resulting in designs of changes not being properly controlled and audited, and the risk of uncontrolled project scope changes.

6.2 SYSTEM DEVELOPMENT

Several interesting and important issues were observed during the system development phase of this project. Of particular note was the difficulty experienced in modelling of the objects, the unique development effort profile, lower project effort and cost, and difficulty in performing integration and testing.

6.2.1 Object Modelling and Design

After initial theoretical OOD/OOP training courses and some practice, the project team found it not too difficult to identify real-world data-carrying objects. But as Jacobson [36] observed, it is more difficult to identify the 'dynamic' objects that describe how the system is used. Other authors state that such objects require no modelling - they simply constitute operations on data-carrying objects and therefore should be included in these objects.

However there are certain behaviours that do not naturally belong to data-carrying real-world objects, and in the team's experience it was better to model separate dynamic objects. Each and every team member experienced some degree of difficulty with the modelling/design of objects. The main reason for this was the very different nature of the object-oriented paradigm as compared to the traditional procedural approach.

The new paradigm required a whole new way of thinking, observing and modelling the world in terms of objects, and ignoring the classroom theory on functional decomposition and modelling the requirements in terms of data-flow diagrams.

Specific difficulties included:

* designing real-world domain objects
* designing dynamic or virtual objects
* how far to breakdown into objects
* defining the 'uses' and 'contains' class relationships

As was observed in this study, the best solution to this problem was a combination of relevant OOD/OOP training, and real experience. (Further review of the training aspects as covered later in this chapter). For systems engineers with average to good C programming experience, it was found that at least 1 to 2 months of programming in an object-oriented environment (such as C++) was an essential prerequisite to any formal OOD/OOP training. After formal training an additional 2 to 4 months was required (on average) before the systems engineer was proficient in OOA/OOD and OOP.

From the project team of 15 members, the re-training period ranged from 3 months for highly skilled senior systems engineers, up to 12 months for junior programmers with minimal previous experience. Table 6.1 below summarises the findings, and indicates the average re-training periods required to acquire solid OOA, OOD and OOP skills.

Junior programmer (1 year programming experience)	12 months
Programmer (2/3 years experience)	7 months
Systems Engineer) (5/6 years experience)	5 months
Senior Engineer (8/10 years experience)	3 months

Table 6.1: Average Object-oriented Re-training Periods

Although the figures in Table 6.1 are not conclusive because of the small single sample, it does suggest that acquiring proficient skills in OOA, OOD and OOP requires a steep learning curve which should not be under-estimated.

6.2.2 Development Effort Profile

In Chapter 3 of this report, the life-cycles of the object-oriented and the traditional development approaches were discussed, and the differences were highlighted. From these models, it was expected that the development effort profiles of the two approaches would be different.

With the relevant project management tools, it was possible to monitor the progress and manpower effort being expended on the object-oriented development. The effort profile of

this new approach did prove to be quite different on the development project.

Figure 6.1 below depicts the effort profiles of the life-cycles of the two approaches. The Object-oriented life-cycle curve is based on the actual man-hours required versus progress measured continuously on this project. This curve closely matched the object-oriented life-cycle as observed by Henderson-Sellers[35].

Since this study did not include a control group to implement the identical project using the traditional approach, the effort profile is based on previous experience and profiles as observed by Lehman[40] and Yourdon[57].

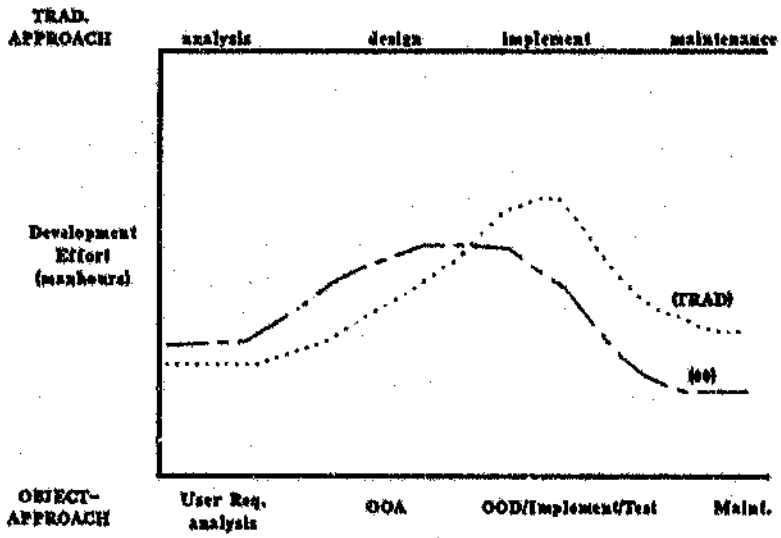


Figure 6.1: Development Effort Profiles of the Two Approaches

The above graph shows some interesting results. The first is that the object-oriented approach requires high initial effort in the analysis/design/implementation phase. This is mainly due to two factors:

(a) The developer has to initially model and implement a large proportion of the objects and its object manager before threads of functionality can be worked and tested.

(b) To benefit from the whole re-use paradigm, greater effort is required to design, . 'ne and reclassify components and component clusters to improve their generality.

The second interesting result is that the effort for the traditional approach as the life-cycle approaches completion of implementation climbs steeply. The effort at this stage is much greater than the object oriented approach. The reasons for this have been well stated by noted authors and have unfortunately been experienced by most organisations practising the traditional approach:

(a) In the object-oriented approach, the detailed design of procedures and data structures is deferred until much later in the development process, and are private to the object. Thus these procedures and data structures are no longer frozen at a high level of system design. Therefore changes at the implementation level are more easily accomplished without requiring changes to the system design itself. This is because object development focuses on data abstraction rather than freezing specific data structures into the object specification.

(b) With the traditional approach, changes in the later part of the life-cycle (and changes there will be), often have a ripple effect right back to the analysis/design phase. The later in the life-cycle the changes are made, the worse the effect.

The third interesting result is the maintenance effort of the two approaches. The ongoing maintenance effort is lower with the object-oriented approach for the same reasons listed above - since maintenance is nothing more than effecting software changes after the main development.

The system life-cycle of the object-oriented and traditional approaches and the related issues have been thoroughly studied and reported by authors such as Booch[13], Meyer[45], Coad and Jourdon[18], Henderson-Sellers[35], Yourdon[58] and Lehman[40]. For the purposes of this study, it is sufficient to note that the results of this study corroborate the findings of these authors, particularly the object-oriented effort profile as observed by Henderson-Sellers. Probably the most important aspect that was evident in this study was that the object-oriented approach required less manpower effort to develop.

Referring to the graph above, the profiles take no account of previous re-use, and the object-oriented approach includes the extensive training and re-training that was required. Discounting extensive training effort and taking advantage of re-use in future projects, it is expected that future projects can be implemented with significantly less effort using the object-oriented component approach to building real-time systems. This fact is further substantiated in the following section which reviews the project effort in greater detail.

6.2.3 Effort, Productivity and Project Cost

One of the primary arguments put forward in this report is that the object-oriented approach results in better productivity and lower total development cost as compared to the traditional procedural approach. Although this study was not intended to include an empirical study on the productivity of the two approaches, the measurements and observations made on this project are noted in this section.

One of the most interesting and comprehensive empirical studies undertaken on productivity and re-use issues of the object-oriented and traditional approaches is that of Lewis et al[41]. In their study, several production variables of the two approaches were scientifically measured and assessed.

Some of the more relevant conclusions reached by Lewis include:

(a) the object-oriented approach substantially improved productivity, although it was believed that a significant part of the improvement was due to the effect of re-use.
(b) software re-use improves productivity irrespective of whether the object-oriented or traditional approach is used,
(c) the object-oriented paradigm has a particular affinity to the re-use process.

In a more recent empirical study by Lewis, Henry, Kafura and Schulman[42], Lewis observed that:

(a) The object-oriented paradigm substantially improves productivity over the procedural approach,
(b) With re-use, the object-oriented paradigm promotes higher productivity than the procedural paradigm,
(c) Software re-use improves productivity no matter what language paradigm is used.

In terms of this study, our findings concur with those of Lewis, except on one point. Lewis believed that their experiment results could not clearly prove productivity improvement with object-orientation when re-use was not a factor. On the development project associated with this study, a definite productivity improvement was noted.

At the start of this project it was believed that the object-oriented paradigm would require high initial effort and that productivity benefits would only be gained with subsequent re-use. However, as development proceeded, it was observed that the object-oriented project (which initially had no benefit of re-use) was implemented with less effort than what would be required for a comparable traditional development.

The extent of the difference was difficult to prove as the study did not have a control group from which to draw empirical measurements for comparison. However, the estimate for productivity improvement (with no reuse) is of the order of 10 to 15 percent, as can be seen from Figure 6.2 below. This in itself in significant when one considers that although the development team attended object-oriented courses as part of the 're-training', this project was the first object-oriented development. For this reason, I believe that it is not unthinkable to achieve a productivity improvement (with no reuse) closer to 20 - 25 percent with fully trained resources.

The reasons for the productivity improvement even with no reuse are twofold:

(a) The particular properties and characteristics of the object-oriented paradigm as described in elsewhere in this study.
(b) In the study it was observed that there is technically no such thing as 'no-reuse benefit' on a first development. Although the development project had no benefit of classes and subsystems developed on previous projects, they were constantly re-using classes and components from the development project itself. In other words, benefits were being gained from re-use on a smaller scale.

Figure 6.2 and Table 6.2 below reviews the development effort (which relates to productivity and cost) for the two approaches compared, with no benefit of re-use, and with benefit of re-use. The development effort figures for the object-oriented approach are based on actual measured data - see Table 6.1 below. However, the data for the traditional approach are estimates only, being based on data from comparable systems previously implemented.

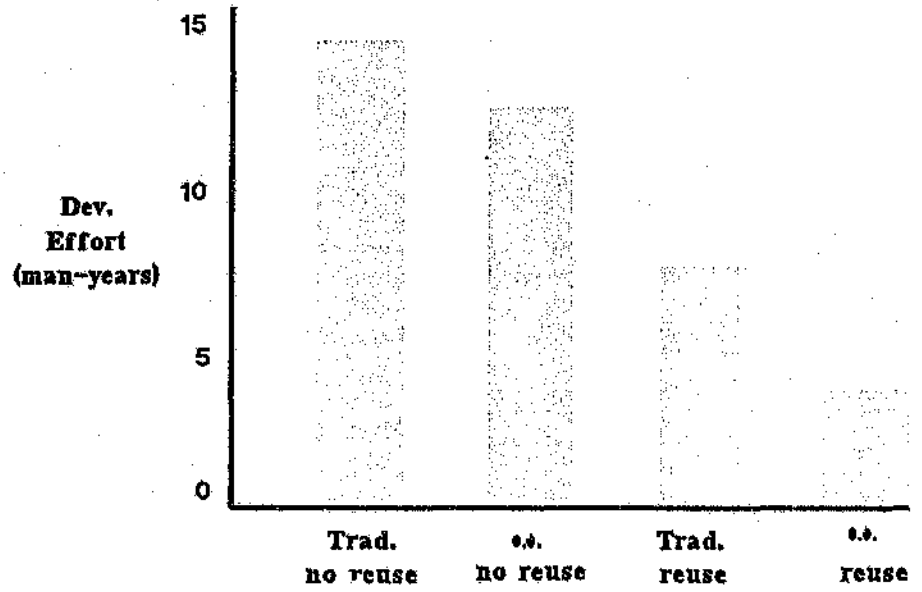


Figure 6.2: Comparison of Development Effort Data

The data for the two approaches with re-use are also estimates. The data for the traditional approach with re-use is based on previously implemented comparable systems in the same telecommunications application domain. The data for the object-oriented approach with re-use is based on the findings of Lewis et al[41], Jrad[30], and personal observations. (I have been involved in Application Consulting for the preparation of proposals for similar applications in the telecommunications application domain).

Jrad et al[30] have done an impressive study by comparing controlled identical developments using the object-oriented and procedural paradigms. In their study, the issues of productivity and performance are empirically compared. It is interesting to note the conclusion that the object-oriented development was developed in half the time the procedural development took. This finding very much endorses the results illustrated in Figure 6.2.

Table 6.1 below details the man-hours actually expended on development of the different subsystem components using the object-oriented approach. The estimates for the man-hours expected for re-using these components in other applications in the application domain are also detailed. (As noted in the previous paragraph, the Application Consulting role in the company has required doing detailed requirements analysis for new applications; identifying components that could be re-used; and estimating manpower required to develop and customise additional components. These new applications have included a fault management application for pressurised transmission cables, and a full network management system for a client managing their own communications infrastructure).

In terms of terminology used in this project, the first column represents development effort measured for the *ObjectView*, *AccessView* and *CustomView* components for the first application in the telecommunications network management domain (ie. no benefit of re-use). The second column represents the estimated development effort for the *CustomView* components required for a new application in the telecommunications network management domain.

BUILDING BLOCK	Development Effort in Man-hours (initial dev.)	Estimated Development Effort (RE-USE)
WMI	2500	1000
MMI	2800	600
RTOMS	2600	1300
HISTORIAN	1500	200
CONFIGURATOR	2200	400
COMMUNICATIONS	2700	0
UTILITIES & other	2800	700
TOTAL MANHOURS	**17100**	**4200**

Table 6.2: Comparison of Development Effort Data

To explain the figures in Table 6.2 in terms of a typical example, the 'pressurised cables' application could be used. For this new application:

* A large portion of the WMI would be re-used - all that needs to be developed is a new WMI device-object for interfacing to the outstation hardware monitoring the pressure breaks. Expected re-usability 60%.
* The MMI would have the largest re-usability, and the only effort required would be to customise a few mimics suited for cables monitoring. Expected re-usability 80%.

* The RTOMS, as described in the previous chapters, would have the lowest level of re-use of all the subsystems. Here the cables' physical network would need to be modelled in terms of related objects in RTOMS. Expected re-usability 50%,
* The Historian and Configurator subsystems would require minimal customisations to cater for particular functionalities particular to the cables environment. Expected re-usability 80 to 85 %
* Communications - a 100% re-usability is expected.
* The Utilities estimated effort caters for development of customised and additional utilities particular to the cables environment.

It will be noted that the development effort estimated for the *CustomView* components required for a new application in the same domain represents approximately 25% of the original development. This represents a re-use of approximately 75%, which is extremely promising.

However, the Building Block Structure depicted in Figure 4.3 of Chapter 4 represents this *CustomView* layer representing approximately 20% of the total system. In retrospect, this was a little idealistic, and the *CustomView* layer is in reality closer to 25%. In other words, re-use in the same application domain is expected to be around 75% instead of the originally envisaged 80%.

In summary of this section, this study has indeed confirmed one of the primary arguments put forward in this study. That is, the object-oriented paradigm results in improved productivity and thus lower development and project costs. More specifically:

* With the traditional approach, re-use gives half the development effort of no re-use,
* Developing new software systems with the object-oriented approach, re-use of object-oriented components requires as little as a third of the development effort as compared to no re-use,
* Contrary to Lewis's[41] findings, this study has observed that even with no re-use, the development effort required for the object-oriented approach is still less than for the traditional approach - mostly due to smaller scale re-use within the project,
* With re-use, the object-oriented approach could potentially result in a development effort one half of that required using the traditional approach. (It is intended to formally prove this in a later study).

6.2.4 Integration and Testing

Although this study argues strongly in favour of the object-oriented component approach, several detracting issues need to be noted. Integration and testing is one of the technical issues that require consideration when adopting the object-oriented approach.

It is necessary to distinguish between low-level component testing undertaken by an individual developer, and integration and system testing undertaken by various members of the development team. Low-level component testing with the object-oriented approach was noticeably easier than testing a similar traditional approach component. This is because the data and functionality are encapsulated within objects, and with clear interfaces to these components, testing of these object components is more directed and deterministic.

For example, the *ObjectView* components of the MMI subsystem were under the development control of a single person. Testing the functionality of, and interaction between the classes such as Display, Motif_widget, Popup, Pulldown, Option, Push_button and Toggle_button proved to be no problem. However, it was the integration and system testing of the components under development of more than one person that proved to be difficult - because of the iterative nature of the development life-cycle of the object-oriented approach. This life-cycle was a continuous iteration of modelling, design, implementation and testing. With the software engineers on the team, at any one time different components were in different phases of the object-oriented development life-cycle.

This made it difficult to synchronise and freeze development activities for integration testing between components. For example, when one subsystem component is ready for integration testing, its associated subsystems could be back in re-modelling or implementation phases. (In the traditional approach, all subsystems reach an integration/testing phase about the same time, and at this stage hopefully all design and implementation has been completed).

Again, an example of this was the MMI subsystem. The developer of the MMI *ObjectView* components and the developer of the MMI *AccessView* components had difficulty in synchronising their development activities so that all required components were in a similar state of readiness to test the *AccessView* fault-handling mimic functionality which required the *ObjectView* display objects. An even more daunting task was to synchronise development activities of the MMI live mimics and the RTOMS subsystem. Both subsystems had to be in a stable 'test' phase before testing of the dynamic links between the live mimics and RTOMS network image could be tested.

In this project, the problem was managed and contained in two ways:

(a) Instead of a single period of integration testing which is characteristic of the traditional approach, several shorter but formal integration tests were planned and scheduled during the entire development cycle of the project. These tests were planned with the involvement and commitment of the development team members, who ensured that their particular subsystem components were in a state fit for integration testing.

(b) A separate integration and testing computer environment was set up, where relevant, 'completed' subsystem components were ported to the testing system. Here these components would be ready for any subsequent formal or ad hoc integration testing with other subsystem components. This allowed developers to continue iterations of remodelling, redesign and prototyping on the development environment. As new versions became available, they would again be ported to the integration computer.

In this project the above technical issue was one of the more unexpected problems associated with the object-oriented approach to building real-time systems. Fortunately, this problem was identified quite early in the project development life-cycle. After adopting a combination of the two measures noted above, the problem was contained and manageable.

Another testing issue that proved to be a problem was that of testing the objects themselves ie. to be able to view attributes and relationships of objects. To overcome this problem the team developed trace and view utilities. For example, for the 'View' utility all classes included a standard 'status enquiry' method which would respond to an external

enquiry on its attributes and current state. The generic 'View' utility would then be used to enquire and report on the attributes of the specified object instance.

6.3 DEGREE OF RE-USE

The major argument put forward and motivated in this study was the need for re-usability of software, and in the earlier chapters of this study the reasons for re-usability were strongly motivated.

For this reason re-usability was the prime focus at all stages of this project, designing and implementing for future re-usability, and re-using what was available to re-use. In Chapter 4 the design of the re-usable component structure was detailed. Components were abstracted laterally according to functional area (eg. WMI, MMI, RTOMS), and abstracted vertically according to the 3-layer component structure (eg. *ObjectView*, *AccessView* and *CustomView*).

In this section where the re-useability is analysed, it is necessary to reflect back on the original designs and philosophies and evaluate the level of success achieved as far as re-use is concerned. Some of the more significant issues that influenced the re-usability during the project are also discussed.

6.3.1 Level of Re-use Achieved

As observed in the previous section, the re-usable *ObjectView* and *AccessView* layer components in reality accounted for approximately 75% of the total application software. That is, 25% of the software would be customised for the specific application. The original expectation was a domain (*ObjectView* and *AccessView*) re-usability of approximately 80%.

The approximate re-usability achieved in reality is shown in a revised Building Block Structure in Figure 6.3 below.

However, it is important to note that the estimates for re-use below are based only on two additional applications that are currently being developed by the project team. It is necessary to corroborate these findings, and it is intended to publish the findings in a future study once further applications have been developed.

Much has been researched and authored about large-scale re-use and the design of re-usable classes and libraries by authors such as Barnes and Bollinger[5], Lewis et al[41], Gibbs et al[27], Meyer[46] and Fischer[24]. However, in many cases this re-use scenario is rather idealistic. Some application domains, such as business/commercial, are well characterised and it is believed that the scenario works well. But in technical real-time systems, where there are very specific requirements, the level of re-use based on stable and static component libraries is not as high.

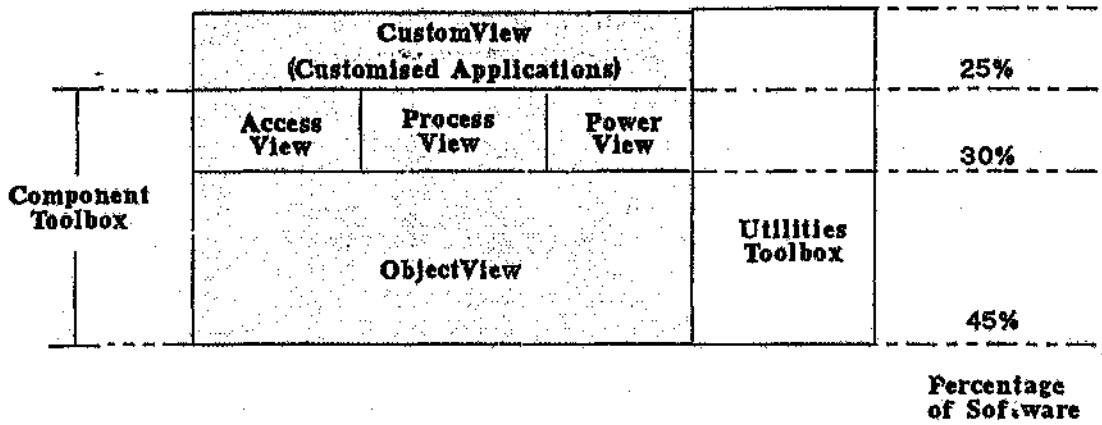


Figure 6.3: Building Block Structure and Re-usability

Furthermore in this study we observed a relatively low degree of 'wholesale' re-use of stable component libraries. As Gibbs et al[27] also observed in their paper 'Class Management for Software Communities', re-usable classes are derived from an iterative process of testing and improvement. Also, because user's requirements are rarely stable, additional constraints and functionalities have to be constantly integrated into existing components.

Anderson[2] in his publication 'Hierarchy Evolution and the Software Lifecycle' was sceptical of wholesale re-use, and observed even domain re-use to be disappointing. He expected the number of new classes introduced per new application (in the same application domain) to slow down, and for the library to become stable, but no such convergence occurred. The re-usability results observed in this study agree with Anderson's observations, and likewise it was also observed that the key to effective use and re-use of domain libraries is to be able to cope with change.

From the results of the project, it can be stated that even at this point the re-use level attained has been very good, and not far off original expectations. The results also concur with those of several other authors who have conducted related studies. These include Meyer[46], Fischer[31] and Lewis et al[41]. Lewis conducted an interesting research which empirically studied the re-usability and productivity issues of object-orientation versus the traditional approach to developing software systems. They observed that the object-oriented paradigm had a particular affinity to the re-use process and that object-orientation yielded no benefit if it was not re-used.

However, observations from this study did not concur with the latter statement of Lewis. Even if no re-use of the object-oriented components were made, the relative ease of maintenance and testing of the implemented system was certainly notable.

A more recent and more extensive empirical study has subsequently been conducted by Lewis, Henry, Kafura and Schulman[42]. They have concluded that:

* Language differences are far more important when programmers re-use than when they do not,
* With re-use, the object-oriented paradigm promotes higher productivity than the procedural paradigm,
* Software re-use improves productivity no matter what language paradigm is used,
* The object-oriented paradigm has a particular affinity to the re-use process.

Although the results from this study require further verification it is believed that the results of this study and other related studies argue strongly in favour of adoption of the object-oriented component approach to building real-time systems. However, it must be noted that to achieve this level of re-use, serious attention was given to several important factors - these are detailed in the subsequent sections of this chapter.

6.3.2 Generality and Size of Re-usable Components

In the project, generality and size were two issues that affected the level of re-use attained:

(a) In terms of generality of applicability; the more general the components were designed and implemented, the lower was the eventual and effective re-usability benefit.

(b) In terms of component size it was observed that components that became too large did not aid the component re-use model. It was observed that components became too large for one of two reasons:

  * the component contained too much *functionality* which resulted in the component being more specific to a particular requirement, and therefore less applicable to other requirements.
  * due to too much *generality* being implemented in the component - making it sometimes unsuitable to re-use.

Biggerstaff and Richter[10] made a similar observation that as the component grows, it also becomes more and more specific, narrowing its application and increasing the cost of using it when modifications are required.

To address the issues of component generality and size, breaking up of the components into smaller components was indicated. However, if components were designed and abstracted too small, the 'hassle' factor in locating, browsing, understanding and using the component exceeded the benefit gained - thus defeating the required objective. During development, there was a concerted effort to make generic components that were not too large.

An example of this in the project was the WAL (World Access Level) object components of the RTOMS subsystem. These objects were initially designed such that each physical indication monitored was represented by a WAL object instance. However, there are many different types of indications, some are single physical bits, while others are complex indications (eg. representing several statuses of that indication).

This resulted in the WAL class becoming very large and also very specific to TNM type indications. The WAL class was then redesigned and broken up so that even coded components of indications were represented by a WAL object. So although many more WAL objects were required to represent the same indications, they were smaller, more general, and thus potentially re-usable in application domains other than TNM. Refer to Appendix F for an example of the base level WAL class definition. This is contained in include file *WAL.h*.

In summary, it was noted in this study that the size and generality of re-usable components had an effect on their level of re-use. It was impossible to devise formal parameters to control the size and generality of components - but an awareness of size and generality considerations of components was required in the development team so that re-use could be optimised.

6.3.3 Shrink-wrap Versus Effective Changing

As observed by Barnes and Bollinger[5], re-use intensive development is best achieved by focusing more on how to *change* software effectively than on how to keep it from changing.

This study proved this observation to be extremely valid. Especially in the technical real-time environment, we found it unwise to try develop excessive component generality at high investment cost, with the specific objective of making components stable, unchangeable and 'shrink-wrapped'. We attempted to develop a component set with moderate generality, but focused more on ease of changing and maintenance of components, since it is reality that changes are always required.

To focus on effective changing was not difficult - the object-oriented design and programming environment provided a far better platform than that provided by the traditional approach. Since changes to components are an aspect of maintenance, these issues are analysed in a later section of this chapter.

On the project, ease of changing components was further enhanced by establishing the following procedures:

(a) Setting up a full software configuration control system (SCCS of UNIX) to control and track changes to components,

(b) Devising a component library structured similarly to the design of the components as detailed in this study - *ObjectView*, *AccessView* and *CustomView*, and within each of those container directories were directories containing components specific to MMI, WMI, RTOMS, Configurator, Historian and Utilities. With this approach, components were relatively easy to locate and retrieve.

(c) Setting up rigorous software development and documentation standards so that components could be understood, copied and modified by anyone on the development team.

6.3.4 Modelling and Functional Abstraction Skills

During the development process of the generic components, it was necessary to constantly ensure adequate inheritance structure, good abstractions and modelling, and to guard against over-specialising classes. As the project progressed, the team improved their object-oriented analysis, design and programming skills, resulting in a re-think on the current designs. This would often result in a re-modelling exercise with restructuring of the class components. Every new improved iteration resulted in better re-use potential of those components.

6.3.5 The Organisational Environment for Re-use

Apart from the technical issues that affected the level of re-use resulting from this project, important organisational issues were also observed. Some of the more significant organisational issues are reviewed in this section, and it is intended to show that without due attention to these factors, it would be difficult to sustain and perpetuate the object-oriented re-use objective in the organisation.

Reviewing the literature from noted authors, specifically Booch[13] and Barnes[5], it was apparent that to realise the full benefits from the object-oriented paradigm, it was necessary for the team/organisation to fully commit to the paradigm. In this study where re-usability was a specific objective, it was ensured that the required equipment, personnel and financial resources were committed to the project. This included:

* proper object-oriented development environment,
* necessary training,
* supply of relevant literature and references and text books,
* setting up of component archive repositories,
* class/component browsing and retrieval environments,
* class/component organisation and configuration control,
* documentation of the components.

Of particular importance was the documentation aspect - specifically the class component documentation that would be required by potential re-users. It was observed that the object-oriented approach relied more on graphical documentation than textual documentation - particularly for documenting the class and object diagrams. For this reason the documentation environment included top-end PCs with ample disc and memory capacity, WordPerfect text-editing environment, the Graphics Gallery and PowerPoint graphics drawing packages, and letter-quality printers with graphics capability.

There is much argument whether re-use is driven by technical factors such programming environment, or by management and organisational factors. A large school of thought sees organisational, management and economic issues as the biggest obstacles to progress in software re-use. Others such as Meyer[46] believe that management factors are over-emphasised, and that it is more important to focus on the technical factors. From the results of this study, it is my opinion that both management and technical factors are equally important, and addressing the one without the other is a sure way to lose out on the full benefits to be gained from the object-oriented approach.

There are many good works and studies on component and class re-usability and management, including that of Gibbs et al[27] that proposes techniques such as class

tailoring, class surgery, class versioning and organisation. However this is a major topic in itself and is not in the scope of this study. Its application to this project is certainly worthy of further study and research.

6.3.6 In Conclusion

In this analysis, the level of re-use attained in this study was described. Issues such as component generalisation and size, managing component change, OOA/OOD skills and the organisational commitment were also reviewed. Each of the above factors affected the ultimate level of re-use.

An important contribution to this study was the level of re-use actually attained on the project, both in terms of re-usable components for new applications as well continuous re-usability within the project itself during development.

With regard to the telecommunications network management domain, an approximate 75% to 80% re-usability of components appears achievable - a very pleasing result. However, several issues had to be addressed, especially the need for a supporting organisational environment.

One disappointment, though not unexpected, was that the component interfaces were not always that clear. The ideal concept of the pure 'software I.C.' which is plug-for-plug compatible with other 'software I.C.s' is really a dream. This study has observed how difficult it is to design and build components that are genuine 'software I.C.s'. However as stated earlier in this chapter, successful re-usability was attained through focusing on ease of changing and maintaining components.

It must be noted that the scope of this study was to build a first TNM application based on the object-oriented component approach. Since then, our organisation has almost completed development of two additional applications in the same TNM domain, and it is from this experience that the 75% re-usability is based. It is intended to refine and further substantiate the re-usability claims as additional applications are built in the future.

6.4 THE 3-LAYERED COMPONENT APPROACH

Before development of the object-oriented components began, much careful thought went into the philosophy and design of the 3-layered *ObjectView*, *AccessView* and *CustomView* approach. In this section, the success of 3-layered philosophy is briefly reviewed.

The arguments relating to the size and abstraction of components (see previous section - Degree of Re-use) may also be used here to motivate the 3-layer categorisation of components.

The 3-layered categorisation provided a practical abstraction of:

* real-time distributed systems generic components (*ObjectView*),
* TNM domain generic components (*AccessView*),
* and application-specific components (*CustomView*).

In practice, the lines dividing these 3 layers were not always clear. Often, object components developed as *ObjectView* components took on TNM domain characteristics and would then become *AccessView* components. Likewise on occasions it was found that *AccessView* components were potentially re-usable in non-TNM domains, and would then be categorised as *ObjectView* components.

A good example of this was the set of objects belonging to the Fault Management component of RTOMS (refer to Figure 5.6). It was initially envisaged that this was very specific to the TNM domain, thus it was an *AccessView* component. However, recent investigation has shown that with very little modification, this component is applicable to other real-time domains, such as industrial process monitoring systems - where process alarms are handled in a similar way. This component was then placed in the *ObjectView* class repository.

Although the dividing lines were not always clear, the 3-layer philosophy created the right 'mindset' in the team and organisation, to optimise for re-usability. In retrospect, this 3-layer abstraction was an excellent design decision. Omitting the *AccessView* or *ObjectView* levels would have resulted in lower re-usability in other application domains, and addition of further levels of domain characterisation would complicate class repositories and retrievals.

Already we are investigating implementing other domain abstractions with re-using the core *ObjectView* components. Of particular interest is the industrial process control/monitoring domain (*ProcessView*) and the electricity reticulation domain (*PowerView*).

This study has therefore proved that the 3-layered approach was another major factor contributing to the success of the object-oriented approach to building real-time systems. It provided a good compromise of practical component abstraction and optimal component re-use.

6.5 MAINTAINABILITY

The Importance of Maintenance

In earlier sections of this chapter the importance of software modifiability was stressed, and for that reason more effort was expended on ease of change rather than 'shrink-wrapping' components. A cogent observation made by Lehman and Belady[40] regarding the maturation of deployed software systems was that 'a program that is used in the real world environment must change or become less and less useful in that environment (the law of continuing change)'.

It is acknowledged that software maintenance is a major activity in most organisations, and as Basset[7] observes, the cost of maintenance is by far the most expensive part the system life cycle. In fact, an extensive study by Wilma Osborne of the National Bureau of Standards in the U.S.A.[47] suggests that 60% to 80% of the total cost of software is due to maintenance.

One of the criticisms levelled at the traditional procedural approach to building software systems is the cost of maintaining the systems. Much has been authored about the maintenance advantages of object-oriented systems, and in this section the practical

maintenance issues as observed in this project are reviewed.

The Maintenance Experience on this Project

This study focused on the first development of the generic building blocks using the object-oriented approach, and as such conclusive metrics on software maintenance effort during the full life cycle are not available. However, there were several changes required during the prototyping, testing and integration of this project - these changes may be viewed as a form of maintenance.

One subsystem on the project that had its fair share of changes, even during development, was the MMI subsystem:

Monitoring of the multiplexer equipment in the transmission systems was a first time for the client and the development team. Much time was spent at technical forums with the client deciding on the graphical symbols and graphical depiction of relationships of the transmission equipment. The consequence was that object components of the MMI had to be regularly modified and the result evaluated with the client. Surprisingly, this was effected easily, and particularly noticeable was the low impact of changes and maintenance on the associated components. This may certainly be ascribed to the encapsulation and data abstraction properties of the object-oriented paradigm.

Analysing the traditional approach, it is well known that the later in the life-cycle changes and maintenance are effected, the more extensive are the ramifications on the associated subsystems' designs and implementations (Lehman[40]). But in this study it was observed that the low maintenance effort experienced so far on the project was primarily due to encapsulation of data and functionality within objects. As noted with the MMI components, changes affected mostly the contents of the objects, in terms of its data or the methods that operate on this data. Particularly noted with the MMI components was that where the classes were well modelled, interfaces to the objects rarely changed, and maintenance had limited effect on the other components.

Maintenance Comparisons of the Two Paradigms

It was not intended to conduct a detailed analysis of the maintenance issues of the two paradigms in this study, as this is an extensive topic in itself. However, Figure 6.4 below depicts proposed maintenance effort profiles of the two approaches for the different phases of the system life-cycle. Here maintenance effort would be defined as the number of man-hours to effect a given unit functional change.

Although these profiles are not based on sturdy empirical data measured in this study or elsewhere - the comparison provides an interesting proposal. It is based on measurements from previous procedurally developed systems, literature readings[6,7,35,40 57], experiences on the object-oriented project development so far, and future expectations.

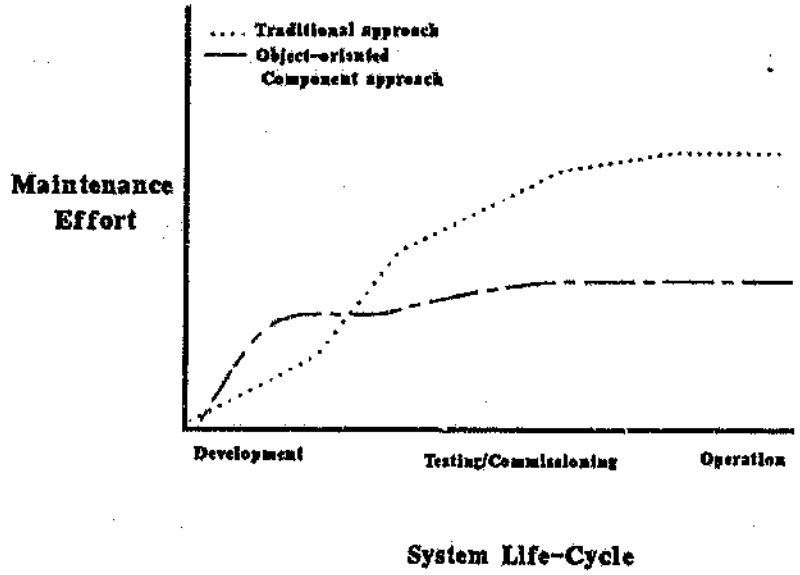


Figure 6.4: Maintenance Effort Comparisons

Referring to Figure 6.4, the following points are noteworthy:

* It has been observed that during early phases of development, modifying object-oriented components requires a little more effort than procedural components. It would be unwise to regard this observation as conclusive at this stage - it may have been due to the project team's unfamiliarity with the object-oriented environment during initial development.
* During the second half of the development, the object-oriented and procedural profiles cross, the procedural effort steadily increasing (even during stable systems operation), and the object-oriented maintenance effort stabilising.

Verification of Lower Maintenance

The development project on which this study is based is currently at the installation phase, and therefore conclusive statistics and metrics on software maintenance effort during the life cycle are not available. However, from the above review the expectation is that the maintenance effort with the object-oriented approach will be considerably better than the procedural paradigm. Therefore the expected maintenance advantages further motivate the adoption of the object-oriented approach for developing real-time software systems.

It is envisaged setting up the mechanisms and procedures so that the long-term full life cycle maintenance metrics can be observed and compared to metrics that exist for the traditional procedural approach.

6.6 THE SOFTWARE BUS

At the start of the project, there were many reservations about implementing or integrating a CMIS communications stack as the basis of inter-process communications. In this section, the results of adopting the CMIS-conformant approach are reviewed.

In this project the software bus philosophy proved to be flexible, versatile and powerful. It provided for easy system extensibility, but also proved to be complicated and initially cumbersome to use.

System Extensibility

One of the most powerful features of the CMIS-conformant bus was the ease with which system functionality could be extended. *Object-manager* (or Agent) processes could be added to the bus as required, as could *Application* processes (refer to earlier definitions). Addition of these processes required no modification or recompilation of the other processes and components. In the worst case, an *Application* could request the services of (or access to) an object which is not contained or registered in any of the *Object-managers* on the network. In this case the relevant application errors are returned to the requester.

On the core project, the MMI, WMI, RTOMS, Configurator and Historian subsystems were implemented as *Object-Managers*, since they need to provide services to external requesters - including other *Object-managers*. However, utilities such as error-log, system-log and watchdog were implemented as *Applications*. Subsequent extensions have proved to be very easy to implement. One example has been trace and debug utilities which were developed. They were developed as *Applications*, developed and compiled separately, and when run merely attach themselves to the software bus and interact with the *Object-managers* (Agents) on the bus.

Further proof of the ease of extensibility was evident with the development of additional report *Applications*. A separate team was given *Application* program templates (a re-usable component in itself), and object dictionaries. Writing a report *Application* was a case of identifying the objects required, which *Object-managers* contained those objects, and writing the relevant SOMI 'GET' requests to those objects. Returned attribute data would then be formatted and displayed or printed as required.

Flexibility

The software bus provided the flexibility because *Object-managers* and *Applications* could be started up, shut-down, modified, and run on other nodes - all in real-time - without affecting other unrelated processes or requiring the entire network to be taken down each time.

Re-use and Productivity

The core *Object-manager* and *Application* process source files served as a 'template' or 'frame' from which new processes were implemented. Therefore productivity improvement was attained due to re-use at this level.

Open Interfacing Platform

An obvious advantage of the CMIS-conformant software bus, although not related to the object-oriented paradigm, is its capability for open interfacing. The entire drive by organisations such as ISO and CCITT (refer to Appendix A) has been to provide industry standards for services and protocols to promote open systems. In the network management domain, the OSI/NM and the CCITT M.30 recommendations specify common 'Q' interfaces between different operating platforms so that multi-vendor equipment can interface at the same level.

As far as the software bus is concerned, it has been designed to be OSI/CMIP/S conformant with a 'Q3' communications interface. Although there has been no immediate need to provide the software bus with the full OSI/CMIP stack, the design of the software bus is such that it can be integrated when required without affecting the applications using the bus.

Performance Issues

There were initially serious reservations about the performance of a software bus which conformed to the CMIP/CMIS services and protocol as proposed by OSI. Because of OSI's objective of generality for Open Systems, the OSI services involved complicated multi-layered communications software. Hence the concern for performance and suitability in a real-time environment.

After several prototyping and benchmarking exercises, a slightly reduced CMIP-conformant software bus was implemented, where not all CMIS services were implemented. The performance figures attained for local and cross-nodal message communications were satisfactory. Actual performance figures are tabled later in this chapter.

Complicated to Use

For the same reasons detailed under the performance issues above, the generality and flexibility provided by OSI's definition for CMIP/CMIS made the software bus complicated to use.

Because the CMIP definition provides for variable argument lists in the service calls (the GET, SET, CREATE, DELETE, EVENT_REPORT and ACTION calls) together with dynamic argument typing, the arguments are in the form of complicated and extensive linked structures. Accessing these calls requires definition of all arguments. This required time-consuming and extensive argument definition for each CMIP service call. The result was low productivity and a high level of syntactic and semantic software errors.

Shortly after this, the CMIP-conformant API (SOMI) was extensively modified to provide a much more user-friendly API. Since then the development team has experienced little problem with using the software bus API. However, it is accepted that a steep learning curve is still required to understand the underlying philosophy and operational concepts of the software bus.

In summary, the CMIP-conformant software bus was one of the major factors contributing to the success of the object-oriented approach to building real-time systems. In particular, the philosophy lent itself to some of our primary objectives of re-usability, flexibility and extensibility, and at the same time providing an open but standard interfacing platform.

6.7 APPLICATION ENGINEERING

As described in an earlier chapter, the project team was divided into three functional roles; the component builder, the application consultant and application engineer. Having come to the end of the first phase of the project, it was necessary to review the validity and practicality of these roles.

In general these defined roles did work, although during the initial development phase of the project the roles of component builder and application engineer were often exercised interchangeably by the same development engineer. As the *ObjectView*, *AccessView* and *CustomView* developments progressed, the roles became more distinct and clear.

Particularly pleasing were two recent occasions where 'Application Engineers' worked closely with an 'Application Consultant' to build TNM application prototypes for client demonstration purposes. Here the Application Engineers made use of the *AccessView* and *ObjectView* components, and developed the relevant prototype *CustomView* components.

These exercises did prove the success of separating the functional roles and also proved how effectively rapid prot .types could be developed using the re-usable components.

6.8 ANALYSIS OF SYSTEM RESOURCES AND PERFORMANCE

In this section the system resources and performance considerations of the object-oriented approach are reviewed. The memory, disc and CPU requirements are analysed, and a comparison is also made with the resource requirements of a comparable traditional system.

System resources and performance are always important parameters in any real-time systems. What was observed was how hungry the object-oriented approach (including the CMIP software bus) was for system resources, particularly memory, disc and processing power.

The object-oriented approach coupled with the CMIS-conformant software bus did utilise more system resources than would be used with a similar procedural approach with conventional inter-process communications. Table 6.3 summarises estimates of the additional resources that were required by the new approach as compared to the conventional approach, expressed as a percentage.

As the study did not cater for a 'control' traditional development, the percentages are based on estimates for a similar development using the traditional approach with conventional inter-process communications. Because both the object-oriented paradigm and the software bus were responsible for additional system resources, Table 6.3 reflects the percentages separately.

RESOURCE	OBJECT-ORIENTED PARADIGM	CMIS-CONFORMANT SOFTWARE BUS
Memory	15%	100%
Disc	10%	10%
MIPS	15%	40%

Table 6.3: Additional Resource Requirements of New Approach

From Table 6.3 it is therefore evident that the object-oriented approach implemented with the C++ programming language utilised 10 to 15% more resources - which is not too significant. However, the use of the CMIP-conformant communications required substantially more memory and processing power.

Why more memory

The object-oriented approach (as applied to real-time systems) requires more memory because each object encapsulates its data (attributes) with its behaviour (object methods). That is, data structures are not shared between 'functions' and there is a certain degree of accepted data redundancy with the object-oriented programming environment. Because of the real-time application environment, the entire real-time image is memory-based, hence the heavy memory usage. In the traditional approach, the real-time image would obviously also be memory based, but data structures would be common and shared, hence more efficient memory usage.

But the CMIP-conformant communications was a very heavy consumer of memory. This was mainly due to *Object-manager* processes that are required to 'hang out' their objects in their containment trees so that its objects are visible to other *Object-managers* and *Applications* via the software bus. It was also necessary for these objects to be memory based.

Why more disc

In terms of disc usage, the object-oriented approach (as experienced on this project) and the CMIP communications also required marginally more disc capacity than the traditional approach would have required.

Why more processing power

Benchmarking and testing of applications proved that the object-oriented programming language used, C++, was particularly efficient and powerful. If good object-oriented programming techniques were followed, stand-alone C++ programs proved to execute marginally faster than their procedural C counterparts. However, in instances where extensive virtual objects were used, the additional de-referencing caused these programs

to be marginally slower than their C counterparts.

However, the CMIP-conformant software bus and the associated communications required far more processing 'horse-power' than a conventional process-to-process communications approach would require.

The main reasons are:

(a) The CMIP-conformant SOMI API calls are very general, with variable argument naming and typing. It was noted that the average CMIP message was six to seven times longer in size the equivalent QCOMM process-to-process message.
(b) The CMIP-conformant protocol suite and services are extensive and complicated, and is a large software subsystem in itself.
(c) The very nature of the CMIP protocol is very message intensive. Any call results in an indication message (going out) and a response message (returned). With scoping and filtering calls, multiple responses are generated. Thus for a given application, the level of software bus traffic is three to four times that of conventional protocol communications.

In terms of disc capacity, the project experienced no serious problems, as the incremental cost of adding disc capacity was less than trying to manage disc utilisation. But problems were experienced with processing power and memory usage, and the development team had to perform several iterations of remodelling and prototyping in order to limit memory usage and CPU utilisation.

Example of Actual Resources Used

The Diagram in Figure 6.5 below is included to illustrate a typical segment of the hardware topology and configuration of the target TNM system that was developed.

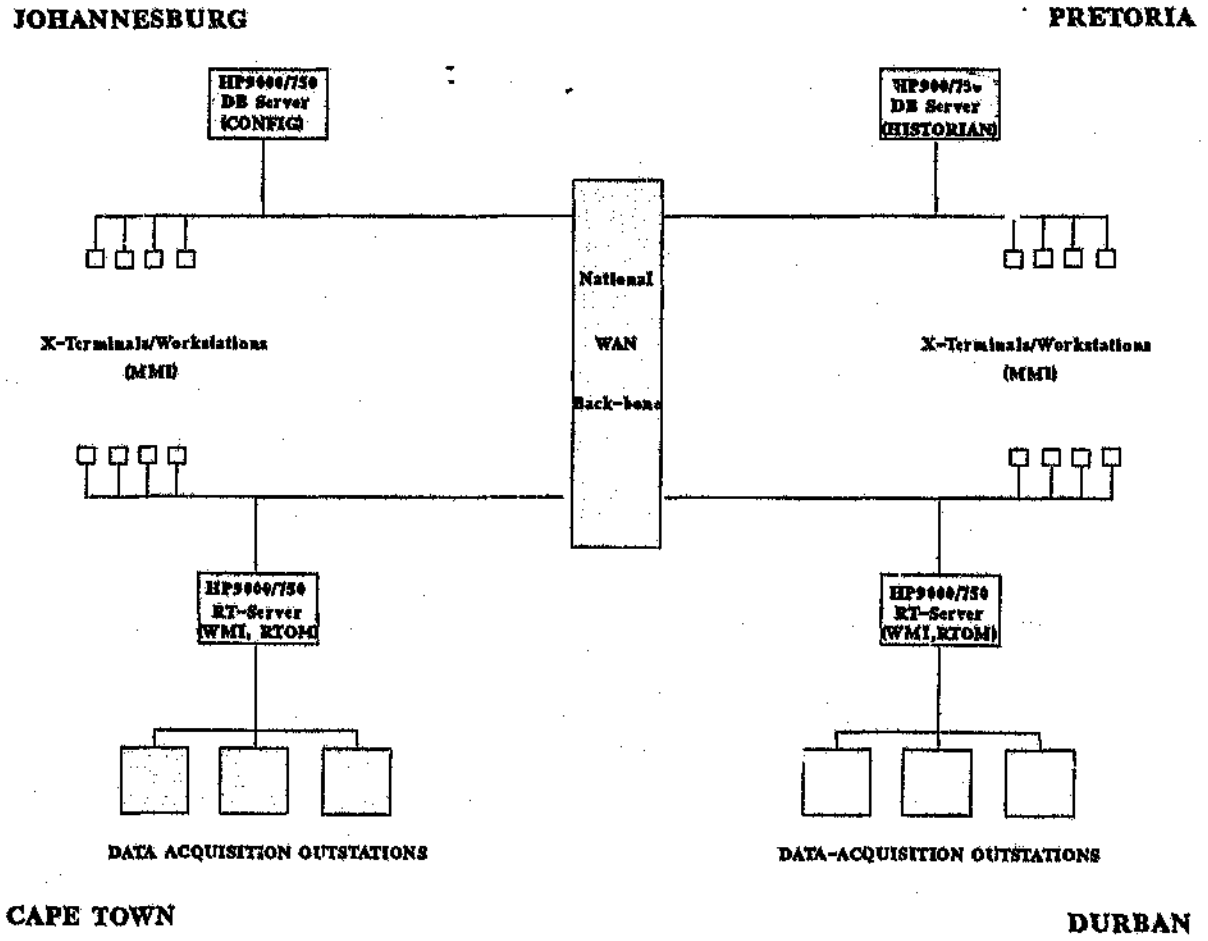


Figure 6.5: Example of TNM Hardware Configuration

With reference to the topology illustrated in Figure 6.5, the Table 6.4 lists the average memory and disc capacities required by the target system of the development project. The memory and disc requirements are expressed on a per object basis. (Note that the figures in the table cannot be used as general guidelines - they are very specific to the telecommunications network management application domain).

Subsystem	Memory (Bytes)	Disc (Bytes)
RTOMS (per configured object)	1209	200
RTOMS (per fault docket)	772	400
RTOMS (per dynamic link)	100	0
MMI (per display object)	950	0
MMI (per dynamic link)	100	0
WMI (per device object)	3000	0
HISTORIAN (per alarm)	0	532
CONFIGURATOR (per object)	0	761

Table 6.4: Memory and Disc Requirements per Object

What has been observed in this study is that the object-oriented approach (together with the CMIS-conformant software bus) in the real-time environment is a heavy consumer of resources. In this project, the problem was not too severe, since the required hardware technologies are available at ever decreasing costs. It was evident that often it was more cost-effective to purchase additional memory, disc and processing resources than to spend man-weeks attempting to optimise resource usage. Nevertheless is was important to monitor the situation, as it was easy for the situation to get out of control.

6.9 MANAGING THE COMPONENT LIBRARIES

In this section, some of the issues relating to the management of component and class libraries are discussed. This is a particularly important issue, because without proper management of component libraries, it is difficult to locate components for re-use and maintenance.

Gibbs et al[87] have defined class management as including:

* class packaging,
* class organisation,
* class retrieval,
* class browsing,
* class maintenance (evolution),
* and class versioning.

In terms of this study it would make sense to extend this definition to refer to components rather than classes, since in this object-oriented approach, the definition of a component could be a single class or a set of related or associated classes. Therefore, in terms of

management, components need to be treated as classes.

In this project, component management was set up to consist of:

* standard documentation of classes,
* development environment standards,
* quality assurance procedures on code and documentation,
* software configuration control and
* component browsing and retrieval tools.

All of the above have been addressed in a satisfactory manner except for the component browsing and retrieval. Rudimentary utilities have been developed for extraction of selectable class information from associated C++ source and header files, but this is seen as a first step.

I am currently evaluating more advanced information systems that would provide full support environments for class and component management. One such environment being investigated is the TeamWork object-oriented CASE environment from Cadre Technologies. The main obstacle is that setting up such an environment requires large financial and organisational commitment.

The team on this project is also investigating developing a component management system, which would include a graphical browser/retriever based on the MMI subsystem re-usable components! The team is using a very interesting paper by Helm and Maarek[34] as the basis of the browser philosophy.

Helm and Maarek[34] acknowledge that the emergence of large collections of re-usable components poses new problems for the software engineer. They propose techniques for browsing amongst functionally related classes, and retrieving classes from libraries based on two sources of information: the source code of each class, and its associated documentation. They have also proposed interesting techniques to meaningfully browse and navigate amongst functionally related components, and integrating information retrieval and browsing tools.

In the project, where the object-oriented approach has been applied to building a TNM application, it was evident that the managing of change was as important as retr l and browsing mechanisms for locating re-usable components. This has also been d by Anderson and Gossain[2], that domain analysis is a continuing process, and the ideal of finally owning stable product-oriented domain-specific libraries is not realistic. Domain libraries are in a constant state of evolution, since there is always a change in the current needs, views and solutions.

This study proved Anderson's observation to be mostly correct, and this had the following effect on this project: as well as developing good mechanisms for browsing, retrieval and documentation of component libraries, it was as important to have good mechanisms to manage change to component libraries.

The problem of class management and related issues is an extensive topic and is beyond the scope of this study. The above review serves only to note the basic issues and highlight the importance of component management. It is also clear that successful class management is paramount to the success of the object-oriented approach, specifically where re-usability and maintainability are concerned.

6.10 PROJECT MANAGEMENT AND CONTROL

Project management and control encompasses many issues. In this analyses, some of the more significant issues relating to project management are highlighted.

In terms of the project scheduling and planning techniques, the techniques practised on this object-oriented approach were no different from the conventional techniques.

Our organisation's Management Information System (MIS) did the tracking and costing of time booked and all financial aspects of this project. Refer to Appendix B for an example of a typical MIS monthly project status report. In conjunction with this, Symantec's PC-based ONTARGET project management package was used. This project management package was used for visual week-by-week tracking of progress, task scheduling and resource planning. Refer to Appendix B for an example of an ONTARGET GANTT chart.

Although these tools worked well for previous traditional systems development projects, they did not work properly with the object-oriented approach. The reason for this is that the input data (specifically time/cost budgets and progress) to these tools were inaccurate.

Two problem areas in project management were identified:

(1) Progress monitoring and management during project development.

(2) The cost/effort estimation for object-oriented systems.

6.10.1 Progress Monitoring and Management

The two above problems are associated in that if progress cannot be accurately monitored, then it is impossible to estimate additional effort required. Or put more simply, if you don't know where you are, then you can't possibly know how much further you have to go.

The cause of these problems is inherent in the very nature of the object-oriented paradigm itself. Unlike the traditional approach's lifecycle which has distinct phases to monitor progress, the object-oriented development life-cycle consists of iterative design, modelling and prototyping phases.

Whereas the progress curve with the traditional life-cycle is ideally near linear, the object-oriented life-cycle progress curve has a sinusoidal characteristic. Figure 6.6 below illustrates the comparative progress curves observed in the study.

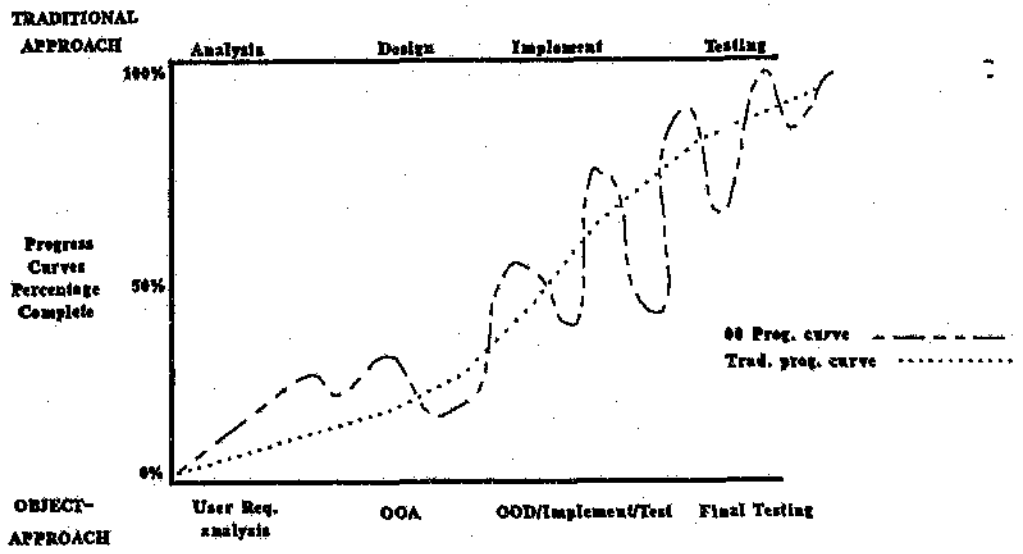


Figure 6.6: Comparative Progress Curves

It was observed that on average at least 3-4 significant remodelling/implementation iterations were required during development of each of the subsystem components. This made it extremely difficult for the development engineers, and even more so for the project manager, to monitor and track the current state of progress.

As an example, after a design, modelling and implementation cycle with the object-oriented approach, the completion status of a component may be estimated at 50%. However, integrating with another subsystem proves modifications are required. This could involve re-design of the classes and object model - which sets progress on that component back to an estimated 35%. As fast as the estimated progress can be retarded due to redesign/remodel, it was observed that it also advances at unreal rates. Because of these characteristics experienced, it was extremely difficult to gauge and report progress.

6.10.2 Cost/Effort Estimation

Accurate effort estimation for the traditional approach is difficult enough, but estimation of effort (manpower) required to implement object-oriented subsystem components was even more difficult.

In the traditional approach, data flow diagrams from the functional analysis could be broken down into functional or process modules. From this breakdown, effort estimations can be made. This same breakdown could be used as the basis of the WBS (work breakdown structure) which is used to control project development and progress.

Based on the same lines, in this study the WBS was composed from a breakdown of the class and object diagrams produced at the early analysis phase. However the results have not been very satisfactory and more accurate and appropriate mechanisms need to be investigated.

6.10.3 In Summary

It is not an understatement that the above project management issues proved to be significant problems with the object-oriented approach. By the time total project implementation was (approximately) 50% complete, I had built up an 'experience' model of where we really were in terms of progress. But this is unsatisfactory, and more formal models for effort and progress management of object-oriented development is required.

Some of the project management issues experienced in object-oriented projects have been researched and reviewed by other authors. Booch[13] proposes that in this evolutionary approach, one can measure progress by counting the classes in a logical design, or the modules in a physical design, that are completed and working. This is a fair statement, but does not take into account that a 'completed' class may not really be 'complete' due to modifications and remodelling inherent in the object-oriented development lifecycle.

Laranjeira[39] has done research on software size estimation for object-oriented systems. It is based on building up a statistical model (which ultimately depends on expert estimations). Du Plessis et al[22] proposes a modification of the Boehm Spiral Model to form a framework for object-oriented software development and management. One of the more detailed works on the subject of metrics for object-oriented systems has been done by Chidamber et al[17]. They have devised an extensive theoretical model for determining a metrics suite for object-oriented design. Other related work has also been undertaken by Goldberg et al[20] and Boehm[12].

However, it is interesting to note that no specific references to these particular progress management problems have been found. Further reading and research on these critical issues is required, and it is expected to publish the findings in a future paper. The author is currently investigating object-oriented progress management and effort estimation along the lines of:

* the average number of design/implementation iterations per class,
* and average effort required to implement a class.

Apart from this it is also necessary to investigate what computer-based tools (ideally suited for object-oriented project management) become available in time to come.

6.11 PERSONNEL TRAINING

In this section the training approach adopted is reviewed and noteworthy problems and issues are discussed.

At the start of the project, none of the development engineers had any experience with object-orientation, C++ programming or with the CMIP software bus concept and protocol. The training was structured in two parts: methodology/technology training, and training in the development team roles.

6.11.1 Methodology and Technology Training

This included training on object-oriented analysis, design, and object-oriented programming in C++.

All members of the development team (except Application Consultants) underwent technology training. Most of this training was conducted by external organisations, but training was backed up with the purchase of several good reference books. The external training was a combination of OOA/D courses run by the University of Witwatersrand Electrical Engineering faculty, and C++ training run by outside consultants.

It was noted that even experienced software engineers struggled somewhat with the concepts of object-orientation. Learning how to model real-world problems in terms of objects, and not fall into the functional decomposition mindset, was difficult. From the experience gained in the study, two conclusions can be made about technology training:

* Miracles should not be expected after a technology course lasting a few days. Much practice was required before development engineers learnt how to truly apply object-oriented principles in their designs and code.

* It became evident that development engineers gained far more benefit from OOA/OOD technology courses if they had at least some exposure to an object-oriented programming environment such as C++ or Smalltalk prior to the courses. The reason for this is that the OOA/OOD courses are focused more on the theory of the paradigm. Previous OOP experience gives students better appreciation of the application and implementation of the object-oriented methodology from a programming point of view.

6.11.2 Development Team Roles Training

The development team roles defined for the project (the toolbuilder, application engineer and application consultant) were described in detail in Chapter 3 of this study. It was an adaption of the roles described by Pliskin[46] and Booch[13]. Pliskin proposed separating software engineer functions from domain engineer functions, and Booch defines the roles of class designers, class implementers and application programmers. Training of the development team was structured around the roles defined for the project.

Component Builder training

Training for Component Builders consisted of:

* philosophy of the building block approach,
* design of the *ObjectView* and the *AccessView* components,
* thorough technology training (OOA/OOD, C++, UNIX)
* use of the CMIP-conformant software bus,
* coding *Object-Managers* and *Applications*.

Application engineer training

This training was focused at Application Engineers and included:

* philosophy of the building block approach,
* using the *ObjectView* and *AccessView* components to build new *CustomView* applications as defined by the Application Consultant,
* use of the CMIP-conformant software bus,
* coding *Object-Managers* and *Applications*,
* integration of the components to build a working s em.

Application Consultant training

It was more important for Application Consultants to have thorough knowledge of their application domains (eg. telecommunications network management), as this type of knowledge is not easily or quickly acquired. Additional training consisted of:

* philosophy of the building block approach
* primary subsystem components
* functionality of these components
* applicability of the components to domain problems

The development team role training was conducted internally by the project managers and the more experienced development engineers. They were run mostly as participative workshops.

Success of the new object-oriented approach to building software systems required full commitment in so far as personnel training was concerned. Training based on the development team roles described earlier proved to work, but special attention was needed to teach object-orientation as a new way of thinking about software systems, and special attention was also required to train all the development engineers on the use of the SOMI software bus.

6.12 OTHER ORGANISATIONAL ISSUES

There are several important organisational issues that were observed in this study. These are described in this section.

6.12.1 Management Commitment to the New Approach

From a management point of view:

(a) There had to be a clear and unambiguous statement of intent of why and how the approach would be adopted.
(b) There had to be a commitment of financial and personnel resources in line with the statement of intent.
(c) The necessary infrastructure had to be set up, such as computer and software development resources.

An important observation made in this study was that there should not be, or perceived to be, a half-hearted approach to adopting the new methodology.

6.12.2 Support and Commitment of Development Staff

Again, to ensure the success of the new approach, it was important to get the support and commitment of the development staff. It should be remembered that people are normally afraid of, and resist change, and if the object-oriented approach is perceived as a radical change that may obsolete their knowledge, the perceived threat can be counter-productive. Chuck Duff and Bob Howard[23] in their article 'Migration Patterns' describe how object-orientation must be introduced as evolutionary and not revolutionary.

To gain the support and commitment of the development staff, it was thus necessary to adopt a phased approach to introducing the object-oriented paradigm to the organisation. The phased approach is discussed in further detail below.

6.12.3 A Phased Adoption of the New Approach

Both Duff et al[23] and Yourdon[56] strongly advocate a phased approach to introducing the object-oriented paradigm to the organisation. In this study, the object-oriented paradigm was not introduced as the 'Silver Bullet', the radical solution to all software's problems. Instead, it was phased in, starting with a small graphics man-machine prototype. Further work and research was encouraged, and initiative was recognised. Once the roots had taken hold in a few individuals, the concepts and philosophies were further driven by project management. In fact, it was so well accepted, that technology decisions were eventually driven by team members themselves.

To review the motivations for a phased approach to adopting the new object-oriented methodology:

(a) It gave our organisation an opportunity to assess the potential benefits and risks.
(b) It allowed our organisation and development staff to gain experience at relatively low risk.

(c) it was less disruptive on the organisation's resources as a whole.
(d) it minimised anxiety because of the safety ties to the existing technology.

It is therefore clear from this and other research studies that a phased approach has many benefits which would assist in a successful transition to the object-oriented paradigm.

6.12.4 Creating a Re-use Culture in the Organisation

One of the major objectives of adopting an object-oriented approach to building real-time systems is re-usability. It could have been quite easy to develop our first 'object-oriented' project as a custom one-off project, but to benefit from re-usability, it was necessary to develop a strong re-usability culture in the organisation.

In this project we did manage to create such a culture, and two factors contributed to this:

(a) **Separating builders from re-users**

This means separating the building of generic components from the reuse and customising of the components. Chuck Duff and Bob Howard[23] in their article 'Migration Patterns' detail how their research has highlighted the importance of separating 'builders' from 're-users'. This was exactly what was done on this project - separating the Toolbuilder role from the Application Engineer function. Separating these roles encouraged the re-use culture. It was also necessary to have regular interaction between these groups - this ensured that Toolbuilders were not developing classes too generic and therefore of limited re-usability, and also ensured that Toolbuilders identified all application requirements that could be modelled generically for reuse.

(b) **The re-usability Champion**

To help drive the organisational awareness of reuse, and to ensure the new approach became a re-usable component approach, it was necessary to have a 'Re-usability' champion. This essentially was one of the author's roles on the project. Under client pressures and deadlines to develop the TNM application, it was easy for the project team, including the Toolbuilders, to focus entirely on the application problem, to the detriment of the reuse objective. It was necessary for the champion to be regularly involved in design and modelling sessions to identify and keep focus on potentially generic object classes and components that could be re-usable in other technical real-time applications.

6.12.5 The Software Porting Mistake

One particular argument that needs to be noted is the folly of 'porting' code from the traditional environment to the new object-oriented environment. By porting is meant trying to fit the procedural flow of traditional programming software into an object model of the OOP environment.

On this project it was attempted to 'port' the original data-acquisition programs (that

interface with the telecommunications outstations) to object-oriented components in the WMI subsystem.

Objects were modelled to fit with the structure of the existing code, but the object methods became too lengthy, complicated and untidy. It was then decided to discard this idea totally and a complete re-model of the WMI component was done, based on the real-world breakdown of the equipment (to be interfaced to), into objects. The results were a great improvement. So the lesson learnt was not to be tempted to 'port' procedural code to an object-oriented environment.

In reviewing the general organisational issues, several important points have been raised, each one a factor in affecting the overall success of this approach. These include the commitment and support of management and development staff, creating a re-use culture in the organisation, and the use of a pilot project as a first introduction to the object-oriented approach. A more technical issue that has been highlighted is that the organisation should not be tempted in allowing code porting from the procedural environment to the object-oriented environment.

6.13 CHAPTER SUMMARY

In this chapter, the results of object-oriented component-based approach to building a real-time TNM application were analysed. This chapter covered many diverse topics associated with adoption of the new approach. Nevertheless, several important notes and observations have been made. Some of the findings challenge those in other related studies, while other observations are corroborated. Notwithstanding some of the pitfalls associated with this approach, the important point of this chapter is that there exists sufficient motivation for the primary arguments of this study.

7. CONCLUSION

In this concluding chapter, the overall objectives of the study are firstly restated. Within context of these objectives, the arguments and contributions of the previous chapters are reviewed, and a final conclusion is presented.

7.1 RESTATEMENT OF THE STUDY'S OBJECTIVES

So that the conclusions and arguments in this final chapter can be viewed in perspective, the objectives of this study are restated below:

* To research and develop an object-oriented component-based approach to building real-time distributed systems.
* To demonstrate the application of this approach by developing a real-time Telecommunications Network Management system,
* To critically analyse the results of the object-oriented approach, specifically from the re-use, cost, maintenance, performance and project management point of view.

7.2 REVIEW OF PREVIOUS CHAPTERS

Chapter 1 introduced the objective of the study, defined the scope of the work and detailed the organisation and outline of the report.

Chapter 2 reviewed the different approaches to building real-time technical software systems, and reviewed the general criticisms levelled at the traditional procedural approach.

Chapter 3 introduced the concept of a component-centred approach, and how this could be achieved with the object-oriented paradigm. Several aspects of the object-oriented approach are detailed, including the implications of software standards and hardware technologies.

Chapter 4 presented the architecture and structure of the generic core building blocks required for real-time systems, and described the components of the various subsystems. The motivation for the software bus was presented and its philosophy and design was described.

Chapter 5 focused on the telecommunications network management application domain and described how the core building blocks were extended to produce a set of domain-specific components or building blocks. Chapter 5 also detailed the customisation and integration of these components to build a real TNM application.

Chapter 6 presented the analysis of the approach, reviewed the arguments, and noted the issues. Specifically, aspects such as re-usability, maintenance, development effort, training and other organisational issues were addressed.

7.3 SATISFACTION OF THE STUDY'S OBJECTIVES

In terms of the objectives set out, the study has been successful. A component-based object-oriented approach to building real-time systems was developed, and from the components were built a real-world telecommunications network management application. The issues and problems associated with this approach were also identified and analysed.

The study has clearly confirmed that three major factors contributed to the success of this approach:

(1) The object-oriented paradigm, which includes the object-oriented analysis, design and programming environments.
(2) The 3-layered component structure philosophy, with the real-time-generic components, the domain-generic components and the customised components.
(3) The CMIS/CMIP-conformant software bus philosophy.

7.4 CONCLUDING REMARKS

The analysis of the approach in Chapter 6 has highlighted the benefits of the object-oriented approach. It has been demonstrated that if the noted issues are addressed, this approach does yield rewarding benefits including genuine re-usability, effective rapid-prototyping, shorter application engineering life-cycles, and improved maintainability and testability.

This study has also shown that this approach can place severe constraints on the organisation. For it to be successful, it requires total commitment from both the development personnel as well as management. The approach requires a very steep learning curve, commitment to relevant training programmes, and availability of suitable development and computer resources.

From a project management perspective, the monitoring and management of development progress proved to be very difficult because of the iterative nature of object-oriented design and software development. Addressing this satisfactorily remains a challenge and is certainly a subject worthy of further research.

Most important, it was observed that to achieve the primary goal of genuine re-usability, it is necessary to instil a 're-usability' culture in the organisation. Contrary to some other studies, the object-oriented approach did not prove to be the 'silver bullet' to all the worlds software problems. As Cox[19] states, the possibility of a software revolution where developers stop programming everything from scratch and begin assembling applications from well-stocked catalogue of re-usable software components, is an enduring, but elusive dream.

I can only agree with the above statement, as well as the comment by Goldberg[20] that components placed in a market face a wide variety of different demands: even well designed components with minimally constrained interfaces will have trouble attracting a critical mass of customers.

Rather re-usable components should be seen within the context of a single organisation, where the re-usable code can become an important business asset, to be treated as an investment and a capital good, rather than simply a cost.

7.5 WHERE TO FROM HERE

In terms of the future, this approach has so far shown such rewarding benefits that I, and our organisation, are fully committed to its further development and application in the real-time systems development arena.

One of the biggest challenges ahead is structuring our organisation to best manage its investment in creating and maintaining re-usable software components. This includes the development and execution of re-use metrics mechanisms, managing practices for design for re-use and design with re-use, library management, best re-use practices, consulting, training and workshops.

Most important is to provide an information systems for component library management, including mechanisms to effectively change and maintain the components.

7.6 FINAL CONCLUSIONS

In summary of this concluding chapter, the study has successfully argued that:

(a) The Object-oriented paradigm proved to be a successful approach to building real-time systems - achieving genuine reusability, lower total development cost, and better maintainability and testability.
(b) The 3-layered structured component approach coupled with CMIP-conformant Software Bus proved very effective in this environment - further improving reusability, maintainability and testability of this approach.
(c) There are several technical and organisational issues that need to be observed when considering this approach.

8. FUTURE RESEARCH

From the content of this report, it is evident that the scope of the subject is extensive. This has resulted in several unanswered questions, and has highlighted areas where further work is required. Chapter 8 concludes the Project Report with a review of these areas requiring further research and study.

8.1 FORMAL METRICS FOR COMPARISON OF THE TWO APPROACHES

Because it was necessary to limit the scope of this study, it was not possible to include the measurement, analysis and comparisons of certain metrics. These include accurate and substantiated metrics for the claimed improvements in productivity, maintenance effort and testability of the object-oriented approach as compared to our organisation's traditional approach to building software systems.

I have already set up mechanisms to monitor these parameters over the life cycle of future applications based on the generic and domain-specific building blocks, and to report the findings in future articles.

8.2 A TOTAL SUPPORT ENVIRONMENT FOR RE-USABILITY

Another area worthy of further study is implementing a total support process for developing generic, re-usable software. As proposed by Gibbs et al[27], Fischer[24], and Anderson[2], this includes designing systems for maintaining and changing the component libraries, defining how to integrate such systems and establishing the appropriate infrastructure to assure wide accessibility of these systems. This should be extended to making re-use and re-design possible, with the design of intermediate abstraction levels being an integral part of the software engineering process.

Much of this can be addressed with use of an information system which will store, manage and categorise libraries or clusters of object class groups, manage a quality control mechanism for classes submitted, and ensure completeness of component/class libraries. Also with a repository, application engineers are more likely to get a standard version rather than a version full of undocumented local modifications.

In fact, on completion of the first phase of the project, it become apparent how important this issue really is. To promote real re-use, a full support environment is required, with tools, help, browsing and information retrieval environments. Our experience confirms the findings of other authors such as Fischer[24], that the cost of finding the required components, understanding them, making changes and re-using must be kept low.

I am currently investigating a whole support environment, using a combination of third-party tools and our own tailored environment, and this could be the subject of a further interesting study.

8.3 MANAGING AND CONTROLLING OBJECT-ORIENTED PROJECTS

Lastly, a critical issue stressed in this study is that of project management techniques required for object-oriented software development projects. Specific areas requiring further research are techniques for effort estimation and formal techniques for measuring and monitoring progress of object-oriented projects.

APPENDIX A: FORMAL AND DE FACTO STANDARDS FOR OPERATING ENVIRONMENTS AND INTER-NETWORKING

In Chapter 3, the new software technologies and standards affecting the re-usable object-oriented approach were briefly reviewed. Of specific interest is the current status of the formal and de facto standards for Open Systems, and the gradual convergence of these two camps.

The subject of standard technologies for Open Systems is extensive and is not in the scope of this study. However this Appendix, which supplements the Chapter 3 discussion on Open Systems, attempts to review the subject in more detail.

A.1 COMMON INTEREST USER GROUPS

Because of frustrations with the continued disagreement between UI and OSF, vendors and other users have banded together to form their own groups. This has primarily been motivated by their need to assert their needs for standards and portability rather than be at the mercy of vendor-created "standards bodies". The following are the more recognised bodies.

International Sector Information Technical Group (IPSIT) - This is an international council trying to harmonise the government OSI profiles (GOSIP's) of its various international members. Its main objective is to encourage a "European Handbook for Open Systems" which defines a common set and OSI interface for technology procurement.

Petrotechnical Open Systems Corporation (POSC) - This is an industry-specific association founded by the oil industry, notably Mobil, Texaco, Chevron, BP and El Aquitaine. Its mission is to standardise on the access, storage and sharing of databases.

User Alliance for Open Systems (UAOS) - This user group's objective is to achieve 'integratable' business information environments. This is to be achieved by an operating system free of vendor affiliation, and believe that even UNIX is still too proprietary. They have joined the Corporation for Open Systems (COS) to support the advance of OSI and integrated services digital network (ISDN) standards.

A.2 CONSORTIUMS

Unlike user groups, consortiums are funded by computer industry hardware and software suppliers. Their motivations are varied and not always in industry's best interest. The motivations to join a consortium include:

* to lead the industry in a technology - to ensure they are not left behind should the technology become a de facto standard
* to evaluate future product development of competitors
* to disrupt progress of the consortium to enable them a marketing edge for a competitive product

The OSF and UI are two of the larger consortiums often competing with each other.

The Open Software Foundation

This foundation was formed to prevent the growing control of UNIX as the Open Operating System by a limited number of vendors. It has released its first operating system OSF/1 which provides for 'standards' operating system, user interface, and distributed computing. Apart from being supported by vendors, government agencies and educational institutions, OSF consists of a software development company and research institute intended to define specifications, and develop products supporting open portable application environments.

Its direction is consistent with the X/Open Common Application Environment, the US National Bureau of Standards Application Portability Profile and other equivalent international standards. It also provides interface compatibility with XPG3 and POSIX specifications meaning that implementations must provide the specified application level interfaces to support portability.

UNIX International (UI)

This is a multi-vendor organisation originating from an earlier alliance between SUN and AT&T in 1988 to develop UNIX, and finds itself competing with OSF to be perceived by users as the standard for Open Systems. Its objective is to direct the evolution of UNIX System V through participation from vendors, software developers, end-users and academic institutions.

UI does not develop any products, rather it defines specifications that can be turned into licensable products by UNIX System Laboratories (USL), a subsidiary of AT&T. Products are developed to comply with IEEE 1003 POSIX standard and the Open Systems Directive issued by the X/Open Group.

A.3 UNIX AS AN OPERATING SYSTEM STANDARD

This development has been centred on an operating environment originally developed by AT&T called UNIX. This operating system, which was originally developed by the academic community, is open in the sense that it is available in a documented form to any software and hardware developers, as well as end-users.

However since the advent of AT&T's UNIX, a considerable number of proprietary versions of the environment have emerged and further standards have evolved.

- 1969 AT&T's UNIX and Open Systems
- 1975 Bell licenses UNIX to Universities (eg. Berkeley)
- 1976 Microsoft first release of UNIX
- 1977 SCO and ISC founded
- 1978 VAX UNIX adopted as university research standard
- 1979 UNIX version 7 from Bell Labs

- 1980 UNIX becomes standard for US Department of Defence DARPA project
- 1982 System V Interface Definition (SVID) developed
- 1983 UNIX System V release 1
- 1986 X Consortium formed
- 1988 UI and OSF formed
- 1990 AT&T creates USL and OSF/1 released
- 1991 MP SVr4.0 released

Unfortunately, the different UNIX standards potentially undermine the fundamental aim of portability. Nevertheless associations of interested parties have been created to steer future development towards a single, standard version of UNIX and other operating environments. These include GOSIP, POSIX and X/Open[31].

A.4 OTHER COMMON APPLICATIONS ENVIRONMENT STANDARDS

There exist several other organisations, either non-profit, educational or driven by government, which define, control and promote operating environment standards. GOSIP, POSIX and X/Open are amongst these.

GOSIP

This is an acronym for Government Open Systems Interconnect Procurement/Profile, and outlines the government policy and strategy for converting to an OSI communications system[31].

POSIX

This is a formal standard relating to Open Systems which assists governments and corporations assessing compliance in the procurement process, and ensuring applications portability. The long-term objective of POSIX is to specify a UNIX-like portable operating system which could well result in creating alternatives and competitors to UNIX and be the start of a new generation of open systems[32]. Originally a project of the USA IEEE Computer Society, in recent years the IEEE's POSIX work has been endorsed by both the American National Standards Institute (ANSI) and the National Institute for Standards and Technology (NIST). It comprises a multitude of IEEE co-ordinated working groups which are defining specifications and standards for user interfaces, networking, multi-processing, security, systems administration, testing, tools, real-time etc. Because POSIX was originally intended to be a generic, non-brand-name specification for UNIX, AT&T's real UNIX will obviously, at least for the moment, become the preeminent POSIX-compliant operating system. However, in the future it may face much of competition in the open systems arena.

X/Open

As discussed previously, X/Open assumes a pivitol role in combining the formal standards and de facto standards created by the market. It is an open international organisation that is evolving a Common Applications Environment by working directly with users, the software industry and hardware vendors. Several governments including the European Community Parliament endorse this body. X/Open's mission is to support the Open Systems movement in order to bring greater value to users while increasing market potential for computer suppliers and independent software vendors. It has formulated the X/Open Portability Guide (XPG) which is a set of specifications combining both formal and de facto standards[31].

A.5 INTERNETWORKING AND PROTOCOL STANDARDS

The more widely used inter-networking and protocol standards can be classified as follows:

Formal Standards	De facto Standards
OSI	TCP/IP
CMIP/CMIS	ARPA/BSD
CMOT	SNMP
OSI/NM	OSF/DME
CCITT/TMN	SNA

TCP/IP

TCP/IP is the de facto standard for inter-networking heterogeneous systems in a network. It originated in the late 1970s as a network technology to link systems in the USA Department of Defence's ARPANET. Because of its longevity and popularity, TCP/IP has acceptance and support amongst almost all major hardware and software vendors, and many applications based on this protocol suite are available. Since the early ARPANET days, TCP/IP has gained new enhancements such as Remote Procedure Calls (RPCs) and Network File System (NFS) to aid development of distributed applications. TCP/IPs strength is that it is one of the few proven mature technologies for inter-operability that works.

TCP/IP is generally regarded as a lightweight solution for managing TCP/IP networks that is easily implemented without large scale resource requirements typically required by OSI implementations. However, it is now accepted that the TCP/IP internet standard can no longer be considered as an interim solution but will continue to be used and coexist with formal standards.

ARPA/BSD

The U.S.A. Defence Advanced Research Projects Agency (DARPA), formerly known as ARPA, is a government agency that began funding in the mid 1970 research with the ARPANET and later the Internet. The Berkeley Software Distribution (BSD) was a version of UNIX that the University of California integrated with the TCP/IP suite of protocols via external funding. This resulted in useful extensions to the basic TCP/IP protocols, including a suite of utility programs as well as 'socket' interface that allowed application programs to access communication protocols.

This is now formally part of the Berkeley 4BSD UNIX distribution, and an abstraction of the services, known as Berkeley Services, has been incorporated into many other vendor's UNIX environments.

SNMP

The Simple Network Management Protocol (SNMP) is the de facto standard for TCP/IP network management, supported by a wide variety of network devices and software. Both TCP/IP and SNMP have been criticised for a number of limitations, such as not addressing all areas of systems and network management in a consistent, complete fashion, and work remains to be done in the area of security.

OSF/DME

The OSF/DME is the Open Systems Foundation's specification for the Distributed Management Environment (DME). It complies with current formal standards and addresses the need to unify the management of systems in heterogeneous environments. Its objective is to support different management protocols and communications stacks, as well as the underlying object models. This it achieves through the use of gateway technologies and Application Programmer Interfaces (API) that translate management protocols and object models, and meet the requirements of both procedure-oriented and object-oriented programming methods.

SNA

This is IBM Corporation's proprietary Systems Network Architecture (SNA) which was first introduced in 1973. It defined the architecture and network products for providing a cohesive communication system in a distributed processing network. It defines the rules and protocols for the interaction of computer, peripheral and software components in a network.

Although SNA's functions are quite similar to the OSI model, the manner in which the functions are implemented are quite different. For this reason it is regarded as too proprietary, and while some regard it as one of the de facto standards, many have questioned this.

OSI

The OSI protocol suite is based on the internationally recognised seven-layer Open Systems Interconnect (OSI) model which was formulated by the International Organisation for Standardisation (ISO). In contrast to TCP/IP, OSI is not yet a product or an established set of protocols. Rather it is a model for applying open standards to help ensure networking computability between heterogeneous systems. Or expressed another way, OSI provides a structure into which internationally agreed standards can be fitted. The OSI Reference Model is a seven-layer framework consisting of the physical, data link, network, transport, session, presentation and application layers. Because of the layered, structured approach, it offers the most global and flexible environment for global networks.

Today, several international governments and organisations have wholly endorsed the OSI standard as the best way to achieve integration of systems and to ensure the widest possible choice of products and suppliers. Examples of this are the U.S.A. and U.K. authorities that have implemented Government OSI Profile (GOSIP) programmes to ensure OSI conformity, and similar support comes the Japanese and European Commission. Many standards organisations such as ANSI (USA) and ETSI (Europe) have adopted OSI for Local and Wide Area Networks.

CMIP/CMIS

ISO has provided a suite of standards for management of distributed systems. They have specified a Common Management Information Protocol (CMIP) and its associated Common Management Information Services (CMIS). CMIP is intended to provide a consistent means of interfacing with a highly varied set of networked resources.

However, CMIP alone does not provide for managing distributed systems. For this reason, standards have also been specified for:

- how management data is organised
- how operations on this data is performed
- and how managed resources can be found in a network

These standards have been partially defined by ISO's Structure for Management Information (SMI) and Guidelines for the Definition of Managed Objects (GDMO). These standards define the conceptual model of how information is to be treated abstractly.

CMIP and the associated standards are based on the concepts of object-orientation and Managed Objects (MO), which is a representation of the resources and services to be managed in terms of its current state (attributes), its behaviour (operations) and the event notifications it may generate. In this way, all operations can be carried out through the same interface and with the same style of interaction - by communicating with objects. The concept of managed objects unifies the seemingly different approaches of systems and network management.

CMIP is the formal standard for OSI management. It is gaining further support and many organisations have incorporated it into their procurement specifications. Already CMIP has been implemented in a variety of environments and on top of several protocol stacks.

CMOT

Because today's de facto standard for networking is the TCP/IP suite of protocols, the upper layers of the OSI protocols have been implemented on top of them as well. An example of this is the Common Management Interface Over TCP/IP (CMOT), which has been specified by the Internet Activities Board (IAB).

OSI/NM

This is OSI's definition of inter-operable interface protocols for Network Management (NM). It is primarily based on existing and draft formal standards from ISO and CCITT. It specifies the protocol elements for network management for use in operations, administration, and providing related communication between different management domains.

CCITT/TMN

CCITT has issued Recommendation M.30 which defines the standards and principles for Telecommunications Management Networks. Within this recommendation, CCITT have defined Recommendation G.773 which defines the protocol suites for "Q" interfaces of transmission systems, and Recommendation G.784 which addresses aspects of the Synchronous Digital Hierarchy (SDH).

The "Q" interface of G.773 supports bi-directional data transfer for the management of telecommunications. It specifies 5 protocol suites for "Q" interfaces: two she. stacks (A1 and A2) for local communications (LAN) and 3 full stacks for either local or wide area communications (LAN or WAN).

The CCITT Recommendation G.784 addresses the management of SDH, including the monitoring and control functions. This recommendation describes a full protocol stack for network management to be carried as an embedded control channel over the SDH data communications channel.

A.6 IN SUMMARY

This Appendix is a brief review of the standards and technologies that had to be considered for this study. The subject of Open Systems and standards is extensive and the reader is encouraged to refer to the references in the Bibliography of this report for more detailed reading.

APPENDIX B: SAMPLE REPORTS FROM PROJECT MANAGEMENT TOOLS

In Chapter 3 the project management methods and tools used on the object-oriented project were reviewed. This appendix contains an example of a Management Information System (MIS) monthly report and an ONTARGET Project Planner GANTT chart as used on this project. For the sake of brevity, the MIS report has been summarised.

Figure B.1 is an example of a typical monthly project management report as obtained from the company Management Information System (MIS).

Figure B.2 is an example of a typical weekly ONTARGET GANTT chart. Note how this project planner package displays task inter-depencies on the GANTT chart, effectively including the primary feature of the PERT chart on the GANTT chart.

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

TOTAL HOURS

WORK	DESCRIPTION	ACTUAL HOURS	BUDGETED HOURS	REVISED BUDGET	ESTIMATE HOURS	TASK STATUS
10A	UTILITIES	185.00	850.00	850.00	850.00	A
10B	SYNTAX CHECK	162.00	162.00	120.00	162.00	C
12A	ASLAN MARKETING	.00			.00	C
12B	ASLAN MARKETING LITERATURE	14.00			14.00	C
12C	ASLAN PRODUCT DEMO PACKAGE	.00			.00	C
13A	ACCESSVIEW PRODUCT SPEC	214.00	1300.00	1300.00	1300.00	A
13B	OBJECTVIEW PRODUCT SPEC	.00	400.00	400.00	400.00	A
13C	ACCEPTANCE TEST PROCEDURE	.00	150.00	150.00	150.00	A
1A	PROJECT MANAGEMENT	826.00	1500.00	1500.00	1500.00	A
1B	PROJECT PLANNING	71.00	100.00	100.00	100.00	A
1C	MEETINGS ON SITE	24.00			77.00	C
1D	TRAVEL	.00			2.00	C
1E	TECHNICAL MEETINGS	176.00			191.00	C
1F	ASLAN PRJ. ADMIN	222.00	500.00	500.00	500.00	A
1G	PQC STANDARD	.00			200.00	A
2A	INTEGRATION	104.00	1760.00	1760.00	1760.00	A
2B	PORT/EVAL/BENCH	81.00	200.00	200.00	200.00	A
2C	SYSTEM ADMIN/MANAGEMENT.	7.00				A
2D	CONFIG DATA TAKE-ON	.00			400.00	A
2E	CONFIG DATA FIX	.00			150.00	A
3B	UNIX/C++ TRAINING	85.00	120.00	120.00	120.00	A
3C	OOD/OOP TRAINING	226.00	400.00	400.00	400.00	A
3D	TOOLKIT ENG. TRAINING	169.00	400.00	400.00	400.00	A
3E	APP. ENG. TRAINING	29.00	400.00	400.00	100.00	A
3F	SALES TRAINING	.00	200.00	200.00	.00	C
4A	MAN-MACHINE INTERFACE (MMI)	1575.00	2000.00	2000.00	4500.00	A
5A	WORLD-MACHINE INTERFACE (WMI)	1399.00	1700.00	1700.00	2400.00	A
6A	REAL-TIME OBJECT MANAGEMENT SYSTEM RTOMS	1854.00	2000.00	2000.00	3600.00	A
7A	CONFIGURATOR	1067.00	2200.00	2200.00	3700.00	A
8A	HISTORIAN	970.00	1400.00	1400.00	1000.00	A
8B	CONFIGURATION REPORTS	.00			100.00	A
8C	HISTORIAN REPORT MANAGER	.00			120.00	A
8D	JSM DAILY REPORTS	.00			400.00	A
8E	DB DOCKET REPORTS	.00			60.00	A
9A	COMMUNICATIONS	548.00	1500.00	1500.00	1032.00	A
		10008.00	19242.00	19200.00	25888.00	

% COMPLETE : 38.66 % OF BUDGET : 52.01 % OF REVISED BUDGET : 52.13

TIME BOOKED DURING PERIOD

INITIALS	HOURS
AB	126.00
BDP	19.00
BH	200.00
CS	39.00
DDB	14.00
GE	175.00
JGBF	207.00

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

JP	90.00
KJL	2.00
LB	71.00
MS	152.00
PBB	228.00
PGF	215.00
PPS	204.00
RDW	1.00
SJ	211.00
TK	47.00
	2001.00

PRODUCTION MEETING NOTES

DATE	NOTES
07-NOV-91	GENERAL: GENERIC GUI - FUNCTIONALLY COMPLETE. CURRENTLY IMPLEMENTING TXVIEW
07-NOV-91	APPLICATION USING THE ACCERSVIEW TOOLBOX
14-NOV-91	% COMPLETE: ONGOING
14-NOV-91	OPERATIONAL PROBLEMS: SYNTAK CHECKER COMPLETE
14-NOV-91	OUTSTANDING BUGS REPORTED NOT FIXED: CORE GUI (MOTIF) TOOLBOX COMPLETE
10-DEC-91	MMI AND WMI COMPONENTS CURRENTLY BEING DEVELOPED. EXPECT TO BEGIN RTOM
10-DEC-91	DESIGN START JANUARY
13-FEB-92	10 JAN 92 RTOMS, COMMS, CONFIG, HISTORIAN DESIGN CURRENTLY IN PROGRESS
02-MAR-92	SOMI VERSION 0.1 COMPLETE, MMI, WMI, RTOM WORK ONGOING. TECHNICAL DEFINITION
02-MAR-92	STILL REQUIRED FOR CONFIG/HISTORIAN.
27-MAY-92	ACCESSVIEW PRODUCT FOR SASNAC, MIMNAC + TESS ABOUT 30% COMPLETE. MMI
27-MAY-92	SUBSYSTEM BEHIND SCHEDULE. NEED RESOURCE HERE, POSSIBLY JJT AT END JUNE.
24-JUN-92	40% COMPLETE - ESTIMATION ON SCHEDULE. CALENDAR MILESTONES UNDER PRESSURE
24-JUN-92	BECAUSE OF MMI.
07-JUL-92	ACCESSVIEW PROJECT ABOUT 42% COMPLETE. MMI AND RTOMS SUBSYSTEMS ARE ON
07-JUL-92	CRITICAL PATH.

PURCHASE ORDERS PLACED

DATE	P ORDERS	S ORD	C#.C#2	QTY	GRV	DESCRIPTION	UNIT PRICE	EXTD PRICE	CANC
15-JAN-92	BDP4236		3.6	1		DELEGATES FOR : PRACTICAL			
15-JAN-92	BDP4236		3.7	7		OBJECT-ORIENTED DESIGN COURSE	945	6615	
15-JAN-92	BDP4240		7.13	5		PHOTCOPY OF ISO DEVELOPMENT			
15-JAN-92	BDP4240		7.14	1		ENVIRONMENT USER'S MANUAL			
26-MAR-92	BDP4434		2.361	1		10M 25PIN MALE TO FEMALE SERIAL PRINTER	33	33	
18-JUN-92	BDP4593		11097.1	1	Y	COLOUR 500C DESK JET CARTRIDGE	97	97	
18-JUN-92	BDP4593		11098.2	1	Y	BLACK 500C DESK JET CARTRIDGE	82	82	
09-JUL-92	BDP4650		11415.0	1	Y	COLOUR 500C DESK JET CARTRIDGE	97	97	
09-JUL-92	BDP4650		11416.0	1	Y	BLACK 500C DESK JET CARTRIDGE	77	77	
							TOTAL AMOUNT OF ORDERS :	6999	

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

GRV'S

DATE	GRV	TYPE	C#.C#2	QTY	PRODUCT	SERIAL	DESCRIPTION	CANCLD
27-MAR-92	BDG6538	GP	2.361	1	CL001		10M 25PIN MALE TO FEMALE SERIAL PRINTER	
29-JUN-92	BDG6752	GL	11321.0	1	CC001		SMP PC	
29-JUN-92	BDG6752	GL	11321.1	1	CC001		64 KB MEMORY	
29-JUN-92	BDG6752	GL	11321.2	1	CC001		2.0 GIGABYTE DAT DRIVE	
29-JUN-92	BDG6752	GL	11321.3	1	CC001		150 MB TAPE STREAMER	
29-JUN-92	BDG6752	GL	11321.4	4	CC001		CPU	
29-JUN-92	BDG6752	GL	11321.5	1	CC001		VGA CARD	
29-JUN-92	BDG6752	GL	11321.6	1	CC001		8 X 4GT	
29-JUN-92	BDG6752	GL	11321.7	7	CC001		8/TC+	
29-JUN-92	BDG6752	GL	11321.8	1	CC001		8/TCFM+	
29-JUN-92	BDG6753	GP	11098.2	1	516261		BLACK 500C DESK JET CARTRIDGE	
01-JUL-92	BDG6760	GP	11097.1	1	516251		COLOUR 500C DESK JET CARTRIDGE	
15-JUL-92	BDG6796	GP	11415.0	1	516251		COLOUR 500C DESK JET CARTRIDGE	
15-JUL-92	BDG6796	GP	11416.0	1	516261		BLACK 500C DESK JET CARTRIDGE	

DELIVERY NOTES

DATE	DN NO	C#.C#2	TYPE	DESCRIPTION	QTY	CUSTOMER ORDER NO	ACCOUNT ID	CANCL
29-JUN-92	BDD6433	11321.0	DN	SMP PC	1			
29-JUN-92	BDD6433	11321.1	DN	64 KB MEMORY	1			
29-JUN-92	BDD6433	11321.2	DN	2.0 GIGABYTE DAT DRIVE	1			
29-JUN-92	BDD6433	11321.3	DN	150 MB TAPE STREAMER	1			
29-JUN-92	BDD6433	11321.4	DN	CPU	4			
29-JUN-92	BDD6433	11321.5	DN	VGA CARD	1			
29-JUN-92	BDD6433	11321.6	DN	8 X 4GT	1			
29-JUN-92	BDD6433	11321.7	DN	8/TC+	7			
29-JUN-92	BDD6433	11321.8	DN	8/TCFM+	1			

SALES ORDERS RECEIVED - INTERNAL

ORD DATE	REG DATE	ORDER	REP	DESCRIPTION	EXT INCOM	EXT COST	INT INCOM	INT COST	MANPOWER	TYPE
16-JUL-92	16-JUL-92	BDS628	AB	ASLAN		-20000	2470000	-1000	1894000	
						-20000	2470000	-1000	1894000	

SALES ORDERS RECEIVED - EXTERNAL

None

COSTING TRANSACTIONS

EXTERNAL INCOME

None

EXTERNAL COST

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

DATE	TRANSNO	DESCRIPTION	TYPE	S	ACC	ACCOUNT ID	AMOUNT
22-JUL-91	BDCBE1264	J.J.T. - EXPRESS SHIPMENT	DEV	T	200		-1194
30-OCT-91	BDCBE1629	S&T - AB	DEV	T	200		-23
14-JAN-92	BDCBE2040	WITS UNIV - OBJECT DESIGN CRSE	DEV	T	200		-6615
03-FEB-92	BDCBE2089	PHOTOCOPY	DEV	T	200		-168
05-MAR-92	BDCBE2238	HP - TRAVEL (WITS)	DEV	T	200		-140
24-APR-92	BDCBE2550	C. LAING - TRAINING	TRG	T	374		-1585
24-APR-92	BDCBE2567	LB - S&T	DEV	T	200		-50
09-JUN-92	BDCBE2642	JGBF - S&T	DEV	T	200		-66
15-JUN-92	BDCBE2662	APPLIED LEARNING TRAINING	DEV	T	200		-1960
16-JUN-92	BDCBE2657	BIFSA - W/SHOP ACCESSVIEW	DEV	T	200		-1764
26-JUN-92	BDCRI2748	DESKJET 500C BLK CART	DEV	T	200	UNIDATA	-82
07-JUL-92	BDCBE2762	ASLAN OUTING	DEV	T	200		-711
21-JUL-92	BDCBE2806	AB - S&T	DEV	T	200		-105
31-JUL-92	BDCRI2889	DESKJET COL CARTRIDGE	DEV	T	200	UNIDATA	-97
							-14560
						GROSS CONTRIBUTION :	-14560

DIRECT RESOURCES - CROSSBILLINGS

DATE	TRANSNO	DESCRIPTION	TYPE	S	ACC	AMOUNT	SUBTOTAL
15-NOV-91	BDCJE532	ASLAN CONTRIBUTION TO MIHNAC	GEN	T	259	300000	
15-JUL-92	BDCJE713	ASLAN RECOVERY FROM IDC J2854	GEN	T	259	195000	
15-JUL-92	BDCJE713	ASLAN RECOVERY FROM IDC J2854	GEN	T	259	70000	565000
31-OCT-91	BDCJE517	DOCUMENTATION XBILLING - OCT91	GEN	T	259	-32	-32
15-JAN-92	BDCJE776	EQIP XBILL 01-JAN-92:31-JAN-92	GEN	T	252	-710	
15-MAR-92	BDCJE782	EQIP XBILL 01-MAR-92:31-MAR-92	GEN	T	*	-710	
15-APR-92	BDCJE784	EQIP XBILL 01-APR-92:30-APR-92	GEN	T	?	-710	
15-MAY-92	BDCJE735	EQIP XBILL 01-MAY-92:31-MAY-92	GEN	T	252	-710	
22-JUN-92	BDCJE775	EQIP XBILL 01-JUN-92:31-JUN-92	GEN	T	252	-710	
15-JUL-92	BDCJE786	EQIP XBILL 01-JUL-92:31-JUL-92	GEN	T	252	-710	-4260
31-JUL-92	BDCJE984	FIN CHARGE 27-JUN-92:31-JUL-92	GEN	T	258	993	993
22-FEB-91	BDCJE404	MAN XBILL weeks 9106:9109	GEN	T	250	-495	
29-MAR-91	BDCJE407	MAN XBILL weeks 9110:9113	GEN	T	250	-2805	
26-APR-91	BDCJE410	MAN XBILL weeks 9114:9117	GEN	T	250	-1400	
28-JUN-91	BDCJE443	MAN XBILL weeks 9123:9126	GEN	T	250	-4322	
26-JUL-91	BDCJE486	MAN XBILL weeks 9127:9130	GEN	T	250	-7136	
30-AUG-91	BDCJE497	MAN XBILL weeks 9131:9135	GEN	T	250	-20747	
27-SEP-91	BDCJE521	MAN XBILL weeks 9136:9139	GEN	T	250	-21854	
01-NOV-91	BDCJE659	MAN XBILL weeks 9140:9144	GEN	T	250	-7510	
29-NOV-91	BDCJE618	MAN XBILL weeks 9145:9148	GEN	T	250	-13979	
27-DEC-91	BDCJE619	MAN XBILL weeks 9149:9152	GEN	T	250	-11340	
31-JAN-92	BDCJE625	MAN XBILL weeks 9201:9205	GEN	T	250	-73404	
28-FEB-92	BDCJE628	MAN XBILL weeks 9206:9209	GEN	T	250	-91979	
27-MAR-92	BDCJE647	MAN XBILL weeks 9210:9213	GEN	T	250	-102286	
01-MAY-92	BDCJE676	MAN XBILL weeks 9214:9218	GEN	T	250	-113664	

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

29-MAY-92	BDCJE679	MAN XBILL weeks 9219:9222	GEN	T	250	-110402	
26-JUN-92	BDCJE708	MAN XBILL weeks 9223:9226	GEN	T	250	-123996	
31-JUL-92	BDCJE976	MAN XBILL weeks 9227:9231	GEN	T	250	-176945	-884264
24-MAY-91	BDCJE351	MISALLOCATION OF EXP CLAIMS	GEN	T	200	-43	
24-MAY-91	BDCJE351	MISALLOCATION OF EXP CLAIMS	GEN	T	200	-800	-843
26-JUN-92	BDNJE521	PETTY CASH - SS	DEV	T	200	-187	-187
29-MAR-91	BDCJE416	PROJ OVERH 23-FEB-91:29-MAR-91	GEN	T	256	-50	
26-APR-91	BDCJE419	PROJ OVERH 30-MAR-91:26-APR-91	GEN	T	256	-50	
31-MAY-91	BDCJE422	PROJ OVERH 27-APR-91:31 MAY-91	GEN	T	256	-50	
28-JUN-91	BDCJE446	PROJ OVERH 01-JUN-91:28-JUN-91	GEN	T	256	-50	
26-JUL-91	BDCJE504	PROJ OVERH 29-JUN-91:26-JUL-91	GEN	T	256	-50	
30-AUG-91	BDCJE507	PROJ OVERH 27-JUL-91:30-AUG-91	GEN	T	256	-50	
27-SEP-91	BDCJE524	PROJ OVERH 31-AUG-91:27-SEP-91	GEN	T	256	-50	
01-NOV-91	BDCJE544	PROJ OVERH 28-SEP-91:01-NOV-91	GEN	T	256	-50	
29-NOV-91	BDCJE554	PROJ OVERH 02-NOV-91:29-NOV-91	GEN	T	256	-50	
27-DEC-91	BDCJE565	PROJ OVERH 30-NOV-91:27-DEC-91	GEN	T	256	-50	
31-JAN-92	BDCJE600	PROJ OVERH 28-DEC-91:31-JAN-92	GEN	T	256	-50	
28-FEB-92	BDCJE633	PROJ OVERH 01-FEB-92:28-FEB-92	GEN	T	256	-50	
27-MAR-92	BDCJE651	PROJ OVERH 29-FEB-92:27-MAR-92	GEN	T	256	-50	
01-MAY-92	BDCJE674	PROJ OVERH 28-MAR-92:01-MAY-92	GEN	T	256	-50	
29-MAY-92	BDCJE723	PROJ OVERH 02-MAY-92:29-MAY-92	GEN	T	256	-50	
26-JUN-92	BDCJE772	PROJ OVERH 30-MAY-92:26-JUN-92	GEN	T	256	-50	
31-JUL-92	BDCJE981	PROJ OVERH 27-JUN-92:31-JUL-92	GEN	T	256	-50	-850
13-MAR-92	BDCJE612	QCOMM SUPPORT XBILLING - FEB92	SUP	T	255	-375	
27-MAR-92	BDCJE680	QCOMM SUPPORT CROSSBILLING	SUP	T	255	-375	
24-APR-92	BDCJE681	QCOMM SUPPORT CROSSBILLING	SUP	T	255	-375	
29-MAY-92	BDCJE682	QCOMM SUPPORT CROSSBILLING	SUP	T	255	-375	
15-JUN-92	BDCJE683	QCOMM SUPPORT CROSSBILLING	SUP	T	255	-375	
20-JUL-92	BDCJE717	QCOMM SUPPORT XBILLING	SUP	T	255	-375	-2250
31-JAN-92	BDNJE444	RECRUITMENT/TRAINING/COS	DEV	T	200	6615	6615
29-MAR-91	BDCJE427	TRAN XBILL 23-FEB-91:29-MAR-91	GEN	T	257	-40	
26-APR-91	BDCJE424	TRAN XBILL 30-MAR-91:26-APR-9	GEN	T	257	-40	
31-MAY-91	BDCJE423	TRAN XBILL 27-APR-91:31-MAY-9	GEN	T	257	-60	
28-JUN-91	BDCJE450	TRAN XBILL 01-JUN-91:28-JUN-91	GEN	T	257	-40	
26-JUL-91	BDCJE508	TRAN XBILL 29-JUN-91:26-JUL-91	GEN	T	257	-55	
30-AUG-91	BDCJE509	TRAN XBILL 27-JUL-91:30-AUG-91	GEN	T	257	-10	
27-SEP-91	BDCJE525	TRAN XBILL 31-AUG-91:27-SEP-91	GEN	T	257	-10	
01-NOV-91	BDCJE545	TRAN XBILL [illegible]-SEP-91:01-NOV-91	GEN	T	257	-60	
29-NOV-91	BDCJE555	TRAN XBILL 02-NOV-91:29-NOV-91	GEN	T	257	-15	
27-DEC	DCJE568	TRAN XBILL 30-NOV-91:27-DEC-91	GEN	T	257	-10	
31-JA	BDCJE463	TRAN XBILL 28-DEC-91:31-JAN-92	GEN	T	257	-202	
28-FEB	JE 78	TRAN XBILL 01-FEB-92:28-FEB-92	GEN	T	257	-69	
27-MAR	JE657	TRAN XBILL 29-FEB-92:27-MAR-92	GEN	T	257	-145	
01-MAY-92	BDCJE688	TRAN XBILL 28-MAR-92:01-MAY-92	GEN	T	257	-122	
29-MAY-92	BDCJE730	TRAN XBILL 02-MAY-92:29-MAY-92	GEN	T	257	-24	
26-JUN-92	BDCJE774	TRAN XBILL 30-MAY-92:26-JUN-92	GEN	T	257	-291	
31-JUL-92	BDCJE983	TRAN XBILL 27-JUN-92:31-JUL-92	GEN	T	257	-461	-1654
15-JUL-92	BDCJE709	XBILL CLI LIC J2926 > J2618	GEN	T	259	157500	
15-JUL-92	BDCJE710	XBILL AV/RT LIC J2931 > J2618	GEN	T	259	75000	

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classiffi ation - INDIRECT INCOME FIN CHARGES: Y

15-JUL-92	BDCJE711	XBILL AV/GUI LIC J2931 > J2618	GEN	T	259	42187	
15-JUL-92	BDCJE712	XBILL AV/DB LIC J2931 > J2618	GEN	T	259	112500	387187
11-MAR-92	BDCJE607	XFER COST OF FLOPPIES	GEN	T	259	-100	-100
						65355	

CONTRIBUTION AND DIRECT RESOURCES : 50795

SUPPLY CROSSBILLING

DATE	TRANSNO	DESCRIPTION	TYPE	S	ACC	AMOUNT
26-JUL-91	BDCJE485	SUPL XBILL 29-JUN-91:26-JUL-91	GEN	T	254	-12
01-NOV-91	BDCJE541	SUPL XBILL 28-SEP-91:01-NOV-91	GEN	T	254	-5
31-JAN-92	BDCJE597	SUPL XBILL 28-DEC-91:31-JAN-92	GEN	T	254	-66
28-FEB-92	BDCJE630	SUPL XBILL 01-FEB-92:28-FEB-92	GEN	T	254	-5
27-MAR-92	BDCJE654	SUPL XBILL 29-FEB-92:27-MAR-92	GEN	T	254	-5
01-MAY-92	BDCJE678	SUPL XBILL 28-MAR-92:01-MAY-92	GEN	T	254	-5
26-JUN-92	BDCJE769	SUPL XBILL 30-MAY-92:26-JUN-92	GEN	T	254	-39
31-JUL-92	BDCJE978	SUPL XBILL 27-JUN-92:31-JUL-92	GEN	T	254	-9
						-146

SALES CROSSBILLING

None

CONTRIBUTION AFTER ALL RESOURCES : 50649

TIMING DIFFERENCES

None

NET PROFIT : 50649

BALANCING TOTAL : 50649

EQUIPMENT

C#.C#2	SERIAL NUMBER	PRODUCT.OPTION	LOC	DESCRIPTION	OWNER	ORD	GRV	DN
2.361		CL001.	CONSUME	10M 25PIN MALE TO FEMALE SERIA	BSW	Y	Y	Y
4.221		51625A.	CONSUME	COLOUR 500C DESK JET CARTRIDGE	BSW	Y	Y	Y
4.222		516261.	CONSUME	BLACK 500C DESK JET CARTRIDGE	BSW	Y	Y	Y
11097.1		516251.	CONSUME	COLOUR 500C DESK JET CARTRIDGE	BSW	Y	Y	Y
11098.2		516261.	CONSUME	BLACK 500C DESK JET CARTRIDGE	BSW	Y	Y	Y
11415.0		516251.	CONSUME	COLOUR 500C DESK JET CARTRIDGE	BSW	Y	Y	Y
11416.0		516261.	CONSUME	BLACK 500C DESK JET CARTRIDGE	BSW	Y	Y	Y

Job : BDJ2618 Description : ASLAN Weeks 9227 to 9231 27-JUN-92 to 31-JUL-92
Owner : AB Job opendate : 21-FEB-91 Close date :
Cost Centre : PROJ Classification : INDIRECT INCOME FIN CHARGES: Y

JOBCOSTING HISTORY

MTH YEAR	PERC COMP	TOT HOURS	EST HOURS	RBUD HOURS	PERC RBUD	SALES ORDERS	TOT INCOME	ACT E COST	GROSS CONTR	DIRECT RESOURCES	NET CONTR	XBILLING	MONTHLY PTD	NET PROFIT
9101		2												
9102		5								-495	-495			-495
9103		22								-3390	-3390			-3390
9104		32								-4880	-4880			-4880
9105		32								-5833	-5833			-5833
9106		60								-10245	-10245			-10245
9107		131								-10245	-10245			-10245
9108	2	333	19210	19080	2			-1194	-1194	-38293	-39487	-12		-39498
9109	21	639	3090	3090	21			-1194	-1194	-60132	-61326	-12		-61337
9110	24	770	3230	3210	24			-1194	-1194	-68663	-69857	-12		-69869
9111	6	880	13920	13920	6			-1217	-1217	225607	224390	-17		224373
9112	7	1007	13920	13920	7			-1217	-1217	214207	212990	-17		212973
9201	13	1786	13920	13920	13			-7832	-7832	147368	139536	-83		139453
9202	19	2786	14520	14520	19			-8000	-8000	55339	47339	-88		47251
9203	21	4041	19088	19000	21			-8140	-8140	-48395	-56535	-88		-56623
9204	28	5421	19088	19000	29			-8140	-8140	-161886	-170026	-93		-170119
9205	34	6636	19330	19200	35			-9775	-9775	-280766	-290541	-98		-290639
9206	41	8003	19330	19200	42			-13565	-13565	-405063	-418628	-98		-418726
9207	39	10008	25888	19200	52			-14560	-14560	65355	50795	-146		50649
MOVE	-3	2005			10			-995	-995	470418	469423	-48		469375

FORECAST

MONTH YEAR	FORE INCOME	FORE E COST	FORE CONTR
PRIOR		-14560	
9208			50649
9209			50649
9210			50649
9211			50649
9212		-1000	49649
9301		-1400	48249
END		-16960	

PROGRESS REPORT

Sales order income :
Actual income : % invoice (to date): % invoice (end of forcast):
Sales order external cost : -20000
Actual external cost : -14560 % cost (to date): 73 % cost (end of forcast): 85
% Manpower 47
% Profit 2

Task Name	Duration	Start	End
TNM PROJECT	1.00 d	02/Jan/1992	02/Jan/1992
MMI SUBSYSTEM	240.00 d	02/Jan/1992	10/Dec/1992
MMI Core Object-manager	25.00 d	02/Jan/1992	06/Feb/1992
OV Graphics Toolkit	50.00 d	02/Jan/1992	13/Mar/1992
OV Picture Editor	30.00 d	16/Mar/1992	24/Apr/1992
OV MMI Editor	60.00 d	27/Apr/1992	06/Jul/1992
OV Text Toolkit	25.00 d	02/Jan/1992	06/Feb/1992
OV Automimics	30.00 d	07/Jul/1992	17/Aug/1992
AV Fault Management	20.00 d	18/Aug/1992	15/Sep/1992
AV Equipment Management	15.00 d	16/Sep/1992	06/Oct/1992
AV Traffic Management	16.00 d	07/Oct/1992	28/Oct/1992
AV MMI Configurator	45.00 d	16/Aug/1992	20/Oct/1992
CV Operational Interface	35.00 d	21/Oct/1992	10/Dec/1992
WMI SUBSYSTEM	188.00 d	02/Jan/1992	25/Sep/1992
WMI Core Object-manager	20.00 d	02/Jan/1992	30/Jan/1992
OV RS-232 Interface Object	10.00 d	31/Jan/1992	13/Feb/1992
OV RS-422 Interface Object	5.00 d	14/Feb/1992	21/Feb/1992
OV QCOMM Interface Object	5.00 d	24/Feb/1992	28/Feb/1992
AV OCTAVE 200 Device Objec	30.00 d	02/Mar/1992	10/Apr/1992
AV OCTAVE 2000 Device Obje	45.00 d	13/Apr/1992	15/Jun/1992
AV RICE-80 Device Object	25.00 d	16/Jun/1992	20/Jul/1992
CV OCTAVE 200 Route Monito	25.00 d	21/Jul/1992	24/Aug/1992
CV OCTAVE 2000 Route Monit	15.00 d	25/Aug/1992	15/Sep/1992
CV RICE-80 Route Monitor	8.00 d	16/Sep/1992	25/Sep/1992
RTOMS SUBSYSTEM	229.00 d	02/Jan/1992	24/Nov/1992
RTOMS Core Object-manager	20.00 d	02/Jan/1992	30/Jan/1992
OV World Access Level (WAL)	16.00 d	31/Jan/1992	21/Feb/1992
OV Logical Level	22.00 d	24/Feb/1992	24/Mar/1992
OV Outlinks	20.00 d	25/Mar/1992	21/Apr/1992
OV Derived Level	26.00 d	22/Apr/1992	28/May/1992
OV Sort Term History	45.00 d	29/May/1992	30/Jul/1992
AV Fault Management	33.00 d	31/Jul/1992	16/Sep/1992
AV Performance Management	28.00 d	17/Sep/1992	26/Oct/1992
AV Equipment Management	20.00 d	27/Oct/1992	24/Nov/1992
CONFIGURATION SUBSYSTEM	186.00 d	02/Jan/1992	22/Sep/1992
OV Database Object Manager	15.00 d	02/Jan/1992	23/Jan/1992
OV SQL-Executor	60.00 d	24/Jan/1992	17/Apr/1992
OV SQL-Generator	50.00 d	20/Apr/1992	26/Jun/1992
OV SQL-Translator	30.00 d	30/Jun/1992	10/Aug/1992
OV Database Services Manage	30.00 d	11/Aug/1992	22/Sep/1992
CV Object-Table Mapping Rule	40.00 d	15/Jun/1992	07/Aug/1992
HISTORIAN SUBSYSTEM	15.00 d	23/Sep/1992	13/Oct/1992
OV Historian Object Manager	15.00 d	23/Sep/1992	13/Oct/1992
UTILITIES	61.00 d	31/Aug/1992	26/Nov/1992
INTEGRATION & FINAL TESTS	53.50 d	11/Dec/1992	02/Mar/1993
DOCUMENTATION	30.00 d	02/Mar/1993	13/Apr/1993

Figure B.2: Example of a Weekly GANTT Chart from the ONTARGET Project Planner

APPENDIX C: DESIGN SPECIFICATION AND IMPLEMENTATION OF THE SOFTWARE BUS

Further technical details on the design and implementation of the SOMI software highway are covered in this appendix. The reader is encouraged to read this appendix if it is required to understand the communication services provided by the CMIP-conformant highway, and the typical API calls available to the application programmer.

C.1 INTRODUCTION

This Appendix provides further technical details on the CMIP/conformant software communications highway used in this object-oriented approach to building real-time software systems.

Firstly, some of the very many terms and abbreviations are defined, and this is followed by a brief overview of the Common Management Information Services/Protocol definition. Our organisation's proprietary TCP/IP communications subsystem, QCOMM, is then described as the transport mechanism for the CMIP-conformant software bus. The Standard Object Management Interface API that was designed is reviewed, and CMIP conformance issues are discussed. Lastly, examples of typical *Object-manager* and *Application* programs are given, and relative performance metrics are tabled.

C.2 GLOSSARY OF TERMS AND DEFINITIONS

DEFINITIONS (as specified in ISO/IEC 9595/9596)

Attribute:
: A property of a managed object (MO).

Common Management Information Service Element (CMISE):
: The particular application-service-element defined in the International Standard.

Common Management Information Services (CMIS):
: The set of services provided by the CMISE.

CMISE-service-provider:
: An abstraction of the totality of those entities which provides CMISE services to peer CMISE-service-users (SOMI, in this case).

CMISE-service-user:
: The part of an application process that makes use of the CMISE

Functional Unit:
: The unit of service used for the negotiation of service options (during association establishment).

Invoking CMISE-service-user:
: The CMISE-service-user that performs a management operation or notification.

Performing CMISE-service-user:
: The CMISE-service-user that performs a management operation or notification invoked by a peer CMISE-service-user.

Standard Object Management Interface (SOMI):
Is the BSW-Data proprietary interface for CMIP/S. The SOMI object encapsulates the functionality provided in a single class. This specific implementation uses QCOMM as the message carrier but provides the mechanisms to change it later to CMOT or even pure CMIP.

SYMBOLS AND ABBREVIATIONS :

ACSE	Association Control Service Element
ASE	Application Service Element
ASN.1	Abstract Syntax Notation 1
AVA	Attribute Value Asertion
CMIS	Common Management Information Service
CMISE	Common Management Information Service Element
CMIP	Common Management Information Protocol
CMOT	Common Management Information Over TCP/IP
CMOQ	Common Management Information Over QCOMM
FTAM	File Transfer Access Management
ISO	International Organisation for Standardisation
MIT	Management Information Tree
MO	Managed Object
OSI	Open Systems Interconnect
PDU	Protocol Data Unit
ROSE	Remote Operations Service Element
SOMI	Standard Object Management Interface
TCP/IP	Transmission Control Protocol over Internet Protocol

CONVENTIONS USED IN CMIP/S PARAMETER DESCRIPTIONS

Conf	Confirm
Ind	Indication
Req	Request
Rsp	Response

M	the parameter is mandatory (=) the value of the parameter is equal to the value passed in the request
U	the use of the parameter is a service-user option - the parameter is not present in the interaction described by the primitive concerned
C	the parameter is conditional

C.3 CMIS/CMIP OVERVIEW

Management information services are used by Manager processes and Agent processes (in this study these are respectively referred to as *Applications* and *Object-managers*) in peer open systems, to exchange information and commands for the purpose of systems and network management.

There are three types of information services provided by CMIS:

(a) an association service;
(b) a management notification service;
(c) a management operation service.

These 3 service categories are reviewed below.

C.3.1 Association Service

Before any two Applications can exchange information, an association between them must be established. During the association establishment phase, various ASEs (such as Manager and Agent processes) may exchange initialisation information to establish an association using ACSE.

The application context specifies the rules required for co-ordinating the information belonging to different ASEs, embedded in ACSE user information parameters. The application context, presentation and session requirements are conveyed using parameters of the M-INITIALISE service.

The M-TERMINATE and M-ABORT services are used for the termination of an association. These may be invoked by either of the CMISE-service-users.

C.3.2 Management Notification Services

The definition and notification and the consequent behaviour of the communicating entities is dependent upon the specification of the managed object which generated the notification and is outside the scope of the CMIS. However, certain notifications are used frequently within the scope of systems management and CMIS provides the following definition of the common service that may be used to convey management information applicable to the notification.

The M-EVENT-REPORT service is invoked by a CMISE-service-user to report an event about a managed object to a peer CMISE-service-user. The service may be requested in a confirmed or a non-confirmed mode. In the confirmed mode, a reply is expected.

C.3.3 Management Operation Services

The definition and notification and the consequent behaviour of the communicating entities is dependent upon the specification of the managed object at which the operation is directed and is outside the scope of the CMIS. However, certain notifications are used frequently within the scope of systems management and CMIS provides the following definition of the common service that may be used to convey management information applicable to the operations.

* The M-GET service: is invoked by a CMISE-service-user to request the retrieval of management information from a peer CMISE-service-user. The service may only be requested in a confirmed mode, and a reply is expected.

* The M-SET service: is invoked by a CMISE-service-user to request the modification of management information by a peer CMISE-service-user. The service may be requested in a confirmed or a non-confirmed mode. In the confirmed mode, a reply is expected.

* The M-ACTION service: is invoked by a CMISE-service-user to request a peer CMISE-service-user to perform an action. The service may be requested in a confirmed or a non-confirmed mode. In the confirmed mode, a reply is expected.

* The M-CREATE service: is invoked by a CMISE-service-user to request a peer CMISE-service-user to create an instance of a managed object. The service may only be requested in a confirmed mode, and a reply is expected.

* The M-DELETE service: is invoked by a CMISE-service-user to request a peer CMISE-service-user to delete an instance of a managed object. The service may only be requested in a confirmed mode, and a reply is expected.

C.3.4 Management Information Tree

Management information may be viewed as a collection of managed objects (instances of different classes of objects), each of which has attributes, and may have defined events and actions (methods). Names of instances of managed objects are arranged hierarchically in a management information tree (MIT).

It is conceivable that there may be dynamic changes to the MIT and that this knowledge may not be instantly available to other open systems.

C.3.5 Managed Object Selection

Managed object selection involves two phases: scoping and filtering.

Scoping entails the identification of the managed object(s) to which a filter is applied.

Filtering entails the application of a set of tests to each member of the set of previously scoped managed objects to extract a subset.

The subset of scoped managed objects that satisfy the scope is selected for the operation.

If no filter is specified, then the set of scoped managed objects is selected for the operation.

Scoping

The base or level zero managed object is defined as the root of the subtree of the MIT from which the search is to commence. Four specifications of scoping level are defined indicating whether the filter is to be applied to:

- the base object alone,
- the n'th level subordinates of the base object,
- the base object and all of its subordinates down to and including the n'th level,

- the base object and all of its subordinates (whole subtree).

Filtering

A filter is a set of one or more assertions about the presence or values of attributes in a scoped managed object. If the filter involves more than one assertion, the assertions are grouped together using logical operators. If a filter test succeeds for a given managed object, then that managed object is selected for performance of the operation.

Synchronisation

CMIS allows a synchronisation parameter to provided a CMISE-service-user to indicate the manner in which operations are to be synchronised across managed object instances when multiple managed objects have been selected by the scope and filter mechanism. The CMISE-service-user may request one of two types of synchronisation: atomic or best-effort. Since the order in which object instances are selected by the filter is a local matter, synchronisation based on order is not meaningful.

C.3.6 Functional Units

The general service capabilities are designated as functional units, where functional units correspond to the support of service primitives or parameters.

All of the CMISE services listed in this Appendix are included in the kernel functional unit. Additional functional units include:

- multiple object selection functional unit,
- filter functional unit,
- multiple reply functional unit.

Further detail on the functional units and the detail description of the CMIS parameters has been omitted from this Appendix for the sake of clarity.

C.3.7 Service Flow

The typical service flow between two processes (a requester application and server/object-manager) is depicted below. These peer-to-peer processes utilise this flow when communication to each other over the software bus.

SERVICE-USER (Application)	SERVICE-PROVIDER (CMIS)	SERVICE-USER (SOMI) (Object-manager)
(1) Req -->	(2) <-- Reject	
	(2) -->	(3) Ind -->
		(4) <-- Rsp
	(5) <-- Reject	
	(5) <--	
(6) <-- Conf		

The Reject is indicated in the confirmation (Conf) call of the CMIS service. A reject is generated for protocol data unit (PDU) type errors and include the following error types:

- duplicate invocation,
- mistyped argument,
- resource limitation,
- unrecognised operation.

Steps (4) to (6) are only performed in the confirmed mode.

Refer to Table C.1 for a summary of the service flows relevant to each of the CMIS services.

Table C.1: Common Management Information Services

C.3.8 Service Definition

The basic CMISE services are listed below.

SERVICE	TYPE
M-ACTION	confirmed or non-confirmed
M-CREATE	confirmed
M-DELETE	confirmed
M-EVENT-REPORT	confirmed or non-confirmed
M-GET	confirmed
M-SET	confirmed or non-confirmed

Parameters returned as part of the confirm primitive may occur as the result of a successful operation or as the notification of an error condition.

Some operations may report an error code. In the event of multiple errors, with one of the errors being a security violation, then the error code "access denied" is returned.

ASSOCIATION SERVICES

Association Establishment

The M-INITIALISE service is invoked by a CMISE-service-user to establish an association with a peer CMISE-service-user. Association establishment is the first phase of any instance of management information service activity.

The parameters of this service are specified by the association-initiator and exchanged when establishing an association. Exchange of this initialisation information is required prior to using management operation and notification services.

Refer to Table C.2 for a summary of the CMIS intrinsic parameters that are defined to be the CMIS specific part of the user information parameter of the M-ASSOCIATE service.

CMIS Intrinsic Parameters

CMIS Activity	Intrinsic	Function	Parameters	Reply
Association Establishment	A-ASSOCIATE	Establish an association with a peer CMIS user	U - Functional Units – Specifies a set of additional Functional Units U Access Control – Permissions U User Information – Application Specific	U U U
Release an association	A-RELEASE	Release a previously established association	None	–
Abort	A-ABORT	Request an abrupt termination of an association	M Abort Source – Indicates the initiating source of the abort The CMIS service provider or CMIS user	M
Report an Event	M-EVENT-REPORT	Report an event to a peer	M Invoke ID – Identifier assigned to this notification M Mode – Request mode (Confirmed or Unconfirmed) M MO class – Class of object in which event occurred M MO Instance – Instance of object M Event Type – Type of event U Event Time – Time of Event Generation U Event Info – Information about the event C Current Time – Time The Response was generated C Event Reply – Reply to the event report C Errors – Error Notification (See error list)	M=I – U U C=I – U C C
Get Attributes	M-GET	Used to retrieve attribute values from a peer Confirmed mode only	M Invoke ID – As before C Linked ID – If multiple replies are to be sent back, then it will have the same value as the invoke ID M Base Object Class – The class of the MO used as a starting point for the selection of the MO on which the filter is to be applied M – Base Object Instance – Specifies the base object instance as for above U Scope – Indicates the subtree, rooted at the base MO, which is to be searched. U Filter – Specifies the set of assertions used to define the filter test to be applied U Access control – Permissions U Synchronisation – How retrievals should be synchronised across object instances U Attribute Identifier List – Set of identifiers for attributes to be returned C MO class – Specifies the class of the MO for which attributes are returned C MO Instance – The instance for which attributes are returned C Current Time – The time at which the response was generated C Attribute List – The set of attribute ID's and values returned C Errors – See list.	M=I C – – – – – – C C U C C
Request Modification of attribute values	M-SET	Used by invoking CMIS user to request the modification of attributes by a peer CMIS user	M Invoke ID – C Linked ID M – Mode of operation (Confirmed or Unconfirmed M Base Object Class – M Base Object Instance – U Scope – U – Filter – U Access Control – U Synchronisation – C MO Class – C MO Instance – M Attribute List – List of attributes and values to be used by performing CMIS user C Current Time – C Errors –	M=I C – – – – – – C C C U C

Table C.2: CMIS Intrinsic Parameters

CMIS Intrinsic Parameters (continued)

CMIS Activity	Intrinsic	Function	Parameters	Reply
Request an action	M-ACTION	Used by invoking CMIS user to request a peer CMIS user to perform an action	M Invoke ID = U Linked ID M = Mode of operation (Confirmed or ?Unconfirmed M Base Object Class = M Base Object Instance = U Scope = U = Filter = U MO Class = U MO Instance = U Access Control = U Synchronisation = U Action Type = Action to be performed U Action Information = Defines the nature, variation or operands for the action U Current Time = U Action Reply = U Errors =	M(=) C C C C C C C C C C C
Create a MO instance	M-CREATE	Used by invoking CMIS user to request a peer CMIS user to create an instance of a MO. Confirmed mode only	M Invoke ID = M MO Class = If the reference object attributes are not supplied then the instance will be initialised using the defaults of the class U MO Instance = If neither the superior object instance nor the MO instance parameters is supplied by the invoking user then the performing user will supply a value U Superior Object Instance = Identifies the existing MO instance which is to be the superior of the new MO instance. If this parameter is supplied then the MO instance parameter shall not be supplied. U Access control = U Reference Object Instance = Specifies an existing object instance of which the attribute values are to be used as defaults for the new instance. U Attribute List = A set of identifiers to be used as defaults for the new instance U Current Time = U Errors =	M(=) C C - - C C C
Delete MO instances	M-DELETE	Used by the invoking CMIS user to request a peer CMIS user to delete a MO instance as well as deregister the instance. Confirmed mode only	M Invoke ID = U Linked ID = M Base Object Class = M Base Object Instance = U Scope = U Filter = U Access Control = U Synchronisation = U MO Class = U MO Instance = U Current Time = U Errors =	M(=) C C C C C C C C C C

Table C.3: CMIS Intrinsic Parameters (continued)

Association Release

The M-TERMINATE service is invoked by a CMISE-service-user to request the orderly termination of an association between peer application entities. The International Standard (CMIS) does not specify any parameters for the M-TERMINATE service.

The M-ABORT service is invoked by a CMISE-service-user to request the abrupt termination of the association between peer application entities.

Refer to Table C.2 for a summary of the intrinsic parameters that are defined to be the CMIS specific part of this service.

MANAGEMENT NOTIFICATION SERVICES

Event Reporting

The M-EVENT-REPORT service is used by a CMISE-service-user to report an event to a peer CMISE-service-user. It is defined as a confirmed and a non-confirmed service. An M-EVENT-REPORT notification is invoked by a CMISE-service-user whenever an event occurs that should be reported to a peer CMISE-service-user.

When the invoking CMISE-service-user has information associated with the occurrence of an event in a managed object, it supplies the identification of the managed object, the time that the event occurred, the type of the event, and other event related information as the arguments to the M-EVENT-REPORT request primitive.

If the event is invoked in the confirmed mode, a result or error primitive is issued by the performing CMISE-service-user to acknowledge its receipt to the invoking CMISE-service-user.

Refer to Table C.2 for the summary of the M-EVENT-REPORT parameters.

MANAGEMENT OPERATION SERVICES

These include the M-GET, M-SET, M-ACTION, M-CREATE, M-DELETE services. Refer to Tables C.2 and C.3 for the summary of the parameters of these services.

Get Management Information

When the performing CMISE-service-user receives an M-GET indication, it validates the semantics of the following parameters (no order implied)

- the base managed object,
- the optional access control information,
- the optional scope information,
- the optional filter information,
- the optional synchronisation information,
- the list of attribute identifiers.

If any parameter is invalid, the operation terminates and the performing CMISE-service-user issues an error response. In the event of multiple errors being detected, with one of the errors being a security violation, the "access denied" error code shall be returned. If no errors are detected, the performing CMISE-service-user attempts to read the value(s) of the attribute(s) requested according to the filter, synchronisation and security parameters, and returns the result(s) and/or error(s) as appropriate.

Set Management Information

When the performing CMISE-service-user receives an M-SET Indication, it validates the semantics of the following parameters (no order implied):

- the base managed object,
- the optional access control information,
- the optional scope information,
- the optional filter information,
- the optional synchronisation information,
- the list of attribute identifiers and values.

If any parameter is invalid, the operation terminates and in the confirmed mode, the performing CMISE-service-user issues an error response. In the event of multiple errors being detected, with one of the errors being a security violation, the "access denied" error code shall be returned. If no errors are detected, the performing CMISE-service-user attempts to modify the value(s) of the attribute(s) requested according to the filter, synchronisation and security parameters, and in the confirmed mode, returns the result(s) and/or error(s) as appropriate.

Management Action

When the performing CMISE-service-user receives an M-ACTION Indication, it validates the semantics of the following parameters (no order implied):

- the base managed object,
- the optional access control information,
- the optional scope information,
- the optional filter information,
- the optional synchronisation information,
- the action type,
- the optional action information.

If any parameter is invalid, the operation terminates and in the confirmed mode, the performing CMISE-service-user issues an error response. In the event of multiple errors being detected, with one of the errors being a security violation, the "access denied" error code shall be returned. If no errors are detected, the performing CMISE-service-user attempts to apply the action to the managed object(s) according to the filter, synchronisation and security parameters, and in the confirmed mode, returns the result(s) and/or error(s) as appropriate.

Create a Managed Object

When the performing CMISE-service-user receives an M-CREATE Indication, it validates

the semantics of the following parameters (no order implied):

- the managed object class and instance,
- the optional superior object instance,
- the optional access control information,
- the optional reference object,
- the list of attribute identifiers and values.

If any parameter is invalid, the operation terminates and the performing CMISE-service-user issues an error response. In the event of multiple errors being detected, with one of the errors being a security violation, the "access denied" error code shall be returned. If no errors are detected, the performing CMISE-service-user attempts to create the managed object , and returns the result and/or error as appropriate.

Delete a Managed Object

When the performing CMISE-service-user receives an M-DELETE Indication, it validates the semantics of the following parameters (no order implied):

- the base managed object,
- the optional access control information,
- the optional scope information,
- the optional filter information,
- the optional synchronisation information.

If any parameter is invalid, the operation terminates and the performing CMISE-service-user issues an error response. In the event of multiple errors being detected, with one of the errors being a security violation, the "access denied" error code shall be returned. If no errors are detected, the performing CMISE-service-user attempts to delete the managed object(s) according to the filter, synchronisation and security parameters, and returns the result(s) and/or error(s) as appropriate.

C.4 USING QCOMM AS THE TRANSPORT STACK

C.4.1 QCOMM Overview

QCOMM is a cross-nodal process-to-process communications subsystem that is proprietary to our organisation, BSW-Data. It is based on the de facto TCP/IP protocol and provides high-performance transparent cross-nodal datagram communication. Its particular features are:

- management and control of resources used by the communications system,
- message buffering and prioritisation,
- guaranteed and confirmed message delivery,
- destination addressing by node number and process name.

Without going into too much detail into the workings of QCOMM, the basic QCOMM API function calls available to application processes are summarised in Table C.4.

READING DATAGRAMS	
Qopen	Opens a primary queue
Qopensecondary	Opens a secondary queue
Qread	Reads a diagram from opened queue
Qclose	Closes a primary queue
Qclosesecondary	Closes a secondary queue

WRITING DATAGRAMS	
Qwrite	Writes a datagram to a queue
Qblockopen	Opens a block write
Qblockwrite	Writes a datagram to the block
Qblockpost	Writes the block to the queue

UTILITY FUNCTIONS	
Gnode	Returns the local node number
Nodesall	Returns a list of all nodes
Nodesup	Returns a list of up nodes
Nodesdown	Returns a list of down nodes
Nametonode	Converts alias name to node
Nodetoname	Returns all alias names for node

Table C.4: QCOMM Application Programmer Interface Functions

C.4.2 QCOMM and SOMI

In the main text of this Report, the reasons and justifications for using QCOMM were put forward. A brief review of these reasons are:

(a) Network management standards are not yet mature and stabilised, particularly those based on OSI's CMIP.
(b) There is already extensive use and investment in TCP/IP networks, and many end-users are sceptical about CMIPs benefits and are unwilling to wait for

OSI-based network management solutions.

(c) For this project, our organisation required a short time to market for competitive reasons.

(d) The costs involved of the OSI-CMIP route: the cost of the stacks, and the costs of thorough integration, testing and optimisation.

(e) Our organisation already had the proven QCOMM product.

For the above reasons, it was decided to develop a CMIS-conformant protocol based on QCOMM, in much the same way as the CMOT protocol, which is the Internet Activities Board (IAB) specified CMIP protocol over TCP/IP. (In this project, this CMIS-conformant protocol was termed CMOQ - Common Management Interface Over QCOMM).

This protocol was then implemented as an API layer which was termed SOMI, the Standard Object Management Interface. SOMI provides for most of the CMIS services and effectively shields application and object-manager processes from the underlying communication protocol. This will enable porting to other communications stacks, such as the full OSI stack, in the future.

C.4.3 Adapting QCOMM for CMIS/CMIP

It was not necessary to make any *major* changes to the existing QCOMM product for use with SOMI. The primary modification required was to make QCOMM event-driven.

The original QREAD call allowed only for reading a queue with wait in which case the receiver process waited on the queue indefinitely until a message arrived. The other mode allowed for QREAD with no-wait, in which case the QREAD call would return immediately if no message was available for reading.

This meant either an indefinite wait or a queue read polling algorithm in application processes. For CMIS conformance and true object-manager (agent) operation, it was necessary for processes to be able to receive messages asynchronously from any requester, in any order, send replies, and still get on with the business of managing and processing its objects in a non-atomic fashion.

For this reason it was necessary to provide event-driven extensions to QCOMM. With these extensions it is possible to define an event queue (as opposed to a normal queue), and define an associated event handler which executes should any message arrive for that queue. In this way a receiver process can define as many event queues as required with never having to explicitly read or poll the queues for messages.

Refer to Figure C.1 for an operational diagram for the Event-driven QCOMM, and refer to Figure C.2 which shows the SOMI/QCOMM layers and interaction.

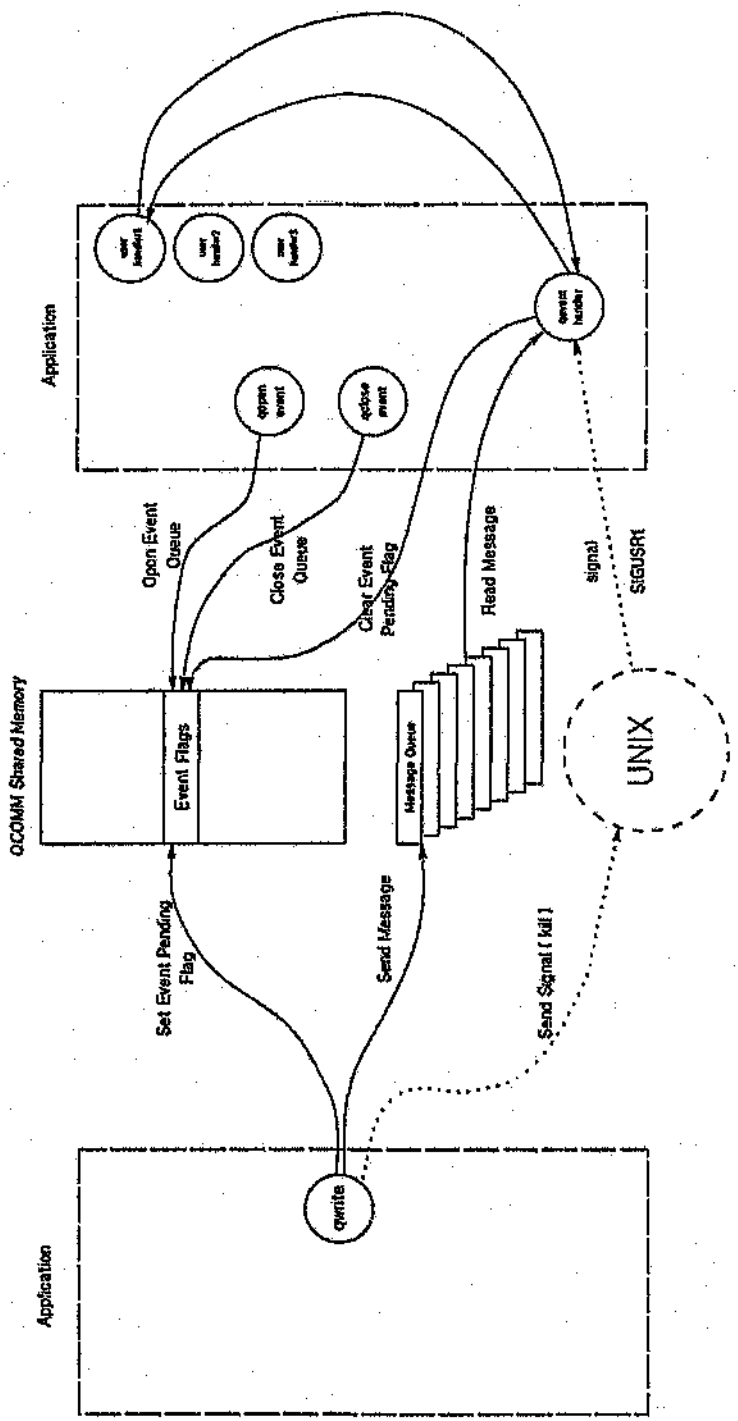


Figure C.1: Event-driven QCOMM Operational Diagram

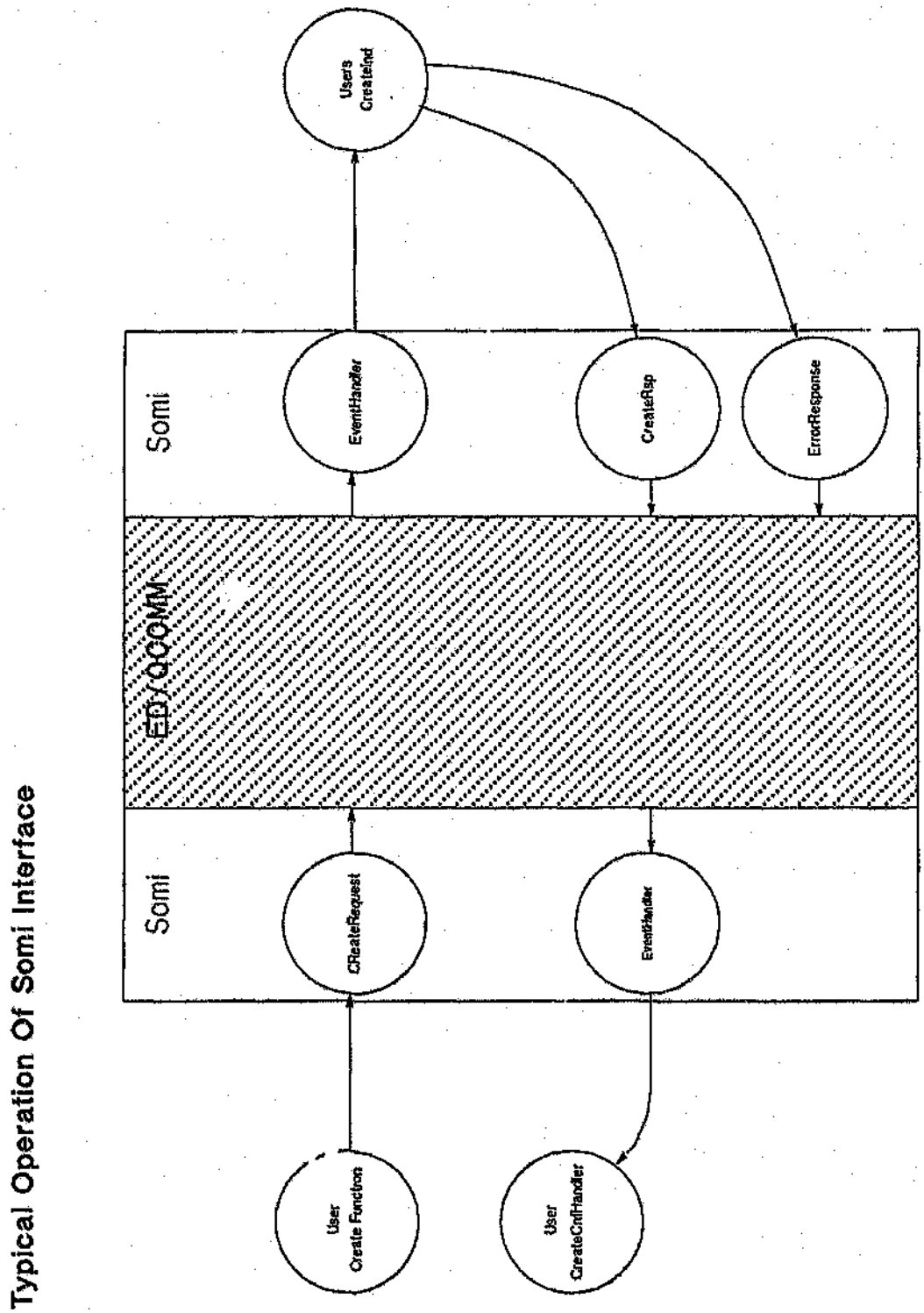


Figure C.2: SOMI/QCOMM Interaction

C.5 SOMI DESIGN AND IMPLEMENTATION

C.5.1 The SOMI Class

The interfacing to the SOMI communication layer is totally handled by and through an object-oriented class called SOMI. The SOMI class is defined as follows:

Association Establishment Methods:

AssAbortReq()
AssAbortRsp()
AssInitialiseReq()
AssInitialiseRsp()
AssTerminateReq()
AssTerminateRsp()

Information Transfer Methods:

ActionReq()
ActionRsp()
CreateReq()
CreateRsp()
DeleteReq()
DeleteRsp()
EventReportReq()
EventReportRsp()
GetReq()
GetRsp()
SetReq()
SetRsp()

C.5.2 The Containment Tree

The containment tree, also known as the naming tree, is a hierarchical structure of managed object instances. It is used to specify the names, or attribute value assertions (AVAs) of the object instances, and their relationships with other object instances.

The SOMI object provides the necessary methods for maintaining the containment tree. There is only one containment tree per application, even if more than one SOMI object is in use. This optimises the CMIS requests should they be directed to an object in the same application but different ASE.

The containment tree is only used in *Object-manager* processes (in other words, only the processes that receive indications), to reflect the containment of the objects that are managed by it. The EVENT_REPORT_IND can be received by any application type (normal or pure object manager or any mix).

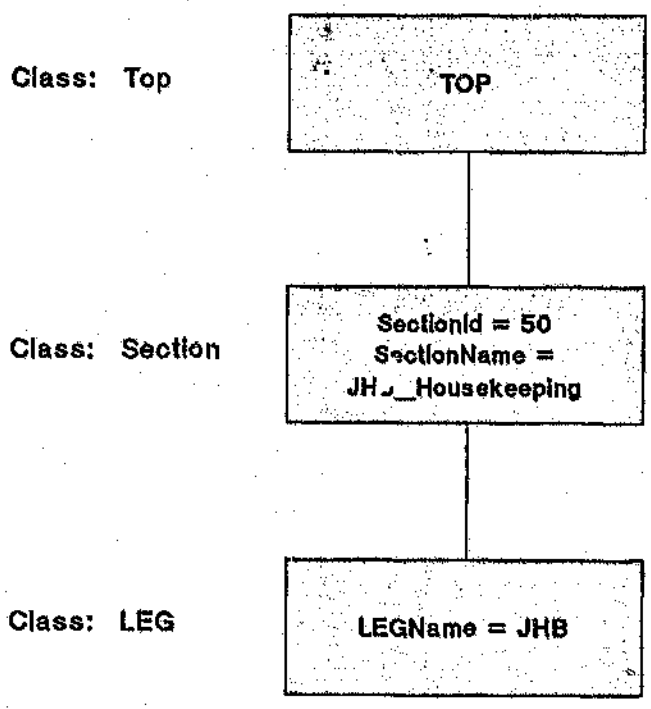
CreateEntry	Creates an entry
FindEntry	Finds an entry
FindSuperior	Finds the superior of an entry

FindSibling	Finds the sibling of an entry (L -> R)
DeleteEntry	Deletes an entry
FindScopedEntry	Finds all entries through the passed scope
GetScopedEntry	Gets all entries previously found with FindScopedEntry

The object naming in the containment tree is done through the use of AVAs and is composed of the attribute name and its value, eg.

"SectionId = 50"
"SectionName = JHB_HouseKeeping"

In this example the specified object uses two AVAs to identify itself, one which is the section identifier, the other one the section name. This makes it possible to find the object with any one of the two attributes (on this level in the containment tree). The object's name on a specific level in the containment tree is known as the Relative Distinguished Name (RDN). To find an object in the containment tree, the full Distinguished Name (DN) is required. The full DN is the concatenation of all the RDNs. For example:



TOP is always the top of the containment tree (per application context). The DN of the LEG instance is either

"SectionId = 50 ; LEGName = JHB"

OR

"SectionName = JHB_HouseKeeping ; LEGName = JHB"

C.5.3 Connection to the Communication Layer

The application programmer uses the Attach method to attach to the communication layer. A parameter passed specifies the name by which this SOMI object is known on the network. This name must be unique per node.

The application can have more than one SOMI object attached to the communication layer (indicating more than one ASE).

The application programmer uses the Detach method to detach the SOMI object from the communication layer. The Detach method cancels all outstanding messages.

Attach	Attach the SOMI object to the network
Detach	Detach the SOMI object from the network

The following utility methods are typically used to determine the name and status of nodes configured on the network:

GetNodeName()	Get the name of a node number
GetNodeNumber()	Get the number of a node name
GetAllNodeNumbers()	Get all the configured node numbers
GetAllUpNodeNumbers()	Get all the up-node numbers
GetAllDownNodeNumbers()	Get all the down-node numbers

Any error number returned from the SOMI communication layer can be converted to a printable string by calling the following method:

ErrorToString()	Convert error to string

C.5.4 Receiving Incoming Messages

The SOMI object uses a callback mechanism to handle all incoming messages, which makes it totally event driven. There is no need for the application to sit and wait in a specific function for data to arrive. This mode of operation makes it feasible to use SOMI in X-Window type applications or applications that must poll an external device.

The programmer can block incoming messages in places where interrupts cannot be handled. The method to do this is:

DisableReceive()	Disable the receiving of messages
EnableReceive()	Enable the receiving of messages

Note: While in a handler function called by the SOMI object, the receiving of messages are automatically disabled until the handler function returns to SOMI.

The application programmer specifies the handlers for each type of CMIS message that is to be received. If no user defined handler is specified, a default handler is called. The default handler will log an error indicating which message was received.

The user defined handler is specified as follows:

SetHandler(HandlerType, UserFunction);

where HandlerType can be any one of:

INDICATION	CONFIRMATION
ASS_ABORT_IND	ASS_ABORT_CONF
ASS_INITIALISE_IND	ASS_INITIALISE_CONF
ASS_TERMINATE_IND	ASS_TERMINATE_CONF
ACTION_IND	ACTION_CONF
CREATE_IND	CREATE_CONF
DELETE_IND	DELETE_CONF
EVENTREPORT_IND	EVENTREPORT_CONF
GET_IND	GET_CONF
SET_IND	SET_CONF

UserFunction is a pointer to a user defined function to handle the specific incoming indication or confirmation.

The SOMI layer keeps track of all incoming messages (*Inds*) that require a *Resp* message. That is done through the use of the *InvokeIdentifier* variable. Combined with the *InvokeIdentifier* the SOMI layer maintains two timeout values, one for the time allowed to issue the first response and a second one for the time allowed between replies (should there be more than one reply - which is indicated in the first reply by the *LinkedIdentifier* variable). These timeout values are originally specified by the request originator. The request is cancelled (ie. an error confirmation send back) should the performing application take longer than what is specified in the timeout values.

C.5.5 Sending Requests

The application programmer uses the *_Req* type methods to send requests to a performing CMISE-service-user (or *Object-manager*). In SOMI the object is addressed either directly or the request primitive is sent via the appropriate manager to the object (if the association is known to the programmer, and then mainly to optimise the access time

to the object). Either way, the association between application entities are maintained by SOMI.

The SOMI layer keeps track of all outgoing requests (*Reqs*) that require a confirmation (*Conf*) message. That is done through the use of the *InvokeIdentifier* variable. Combined with the *InvokeIdentifier* the SOMI layer maintains two timeout values, one for the time allowed to wait for the first confirmation and a second one for the time allowed between confirmations (should there be more than one - which is indicated in the first confirmation by the *LinkedIdentifier* variable). These timeout values are specified as parameters in the *Req* method. The request is cancelled (ie. an error confirmation send back) should the performing application take longer than what is specified in the timeout values.

C.6 SOMI API CALLS

Table C.5 indicates which SOMI methods are the equivalent of the associated CMIP/S service:

CMIP/S	SOMI API
Association Establishment:	
M - ABORT	AssAbortReq(), AssAbortRsp()
M - INITIALISE	AssInitialiseReq(), AssInitialiseRsp()
M - TERMINATE	AssTerminateReq(), AssTerminateRsp()
Information Transfer:	
M-ACTION	ActionReq(), ActionRsp()
M-CREATE	CreateReq(), CreateRsp()
M-DELETE	DeleteReq(), DeleteRsp()
M-EVENT-REPORT	EventReportReq(), EventReportRsp()
M-GET	GetReq(), GetRsp()
M-SET	SetReq(), SetRsp()

Table C.5: SOMI Methods Equivalent to CMIP/S Services

C.7 OBJECT MANAGERS AND APPLICATIONS

New object-manager processes and application processes can be created by building onto templates which we created. These frames contain the default composition, services and options for the application domain.

With CMIP/CMIS conformance and object-orientation there are two types of processes or programs; *Object-managers* and *Applications*. *Object-managers* can be likened to 'Server' or 'Agent' processes which understand incoming requests from remote or local 'Client' processes. *Applications* can be likened to these 'Client' or 'Manager' processes from which the requests originate.

In terms of CMIS however, the major difference between *Object-managers* (OMs) and *Applications* is that the *Object-manager* contains 'Managed Objects' (MOs), and *Applications* do not. With an *Object-manager*, all MOs within its control are registered within the *Object-manager's* containment tree. This means that any other *Object-manager* or *Application* may address requests at any OM's objects directly.

However, *Applications* do not have SOMI objects under its control, and can therefore receive no external requests to its objects. *Applications* can only initiate requests. A good example of a SOMI *Application* is a report program which would request attribute data from objects in the Historian *Object-manager* (which would be a SOMI *Object-manager*) and display the data in a formatted report.

Both *Object-managers* and *Applications* have to attach themselves to the SOMI Bus before they can service or initiate any operations. It should be noted that *Applications* are implemented in an object-oriented manner with classes and objects, but because it has no SOMI containment tree, its objects are internal only and are not externally visible to other *Applications* or *Object-managers*.

Figure C.3 depicts an example pseudocode listing of an *Object-manager* frame, and Figure C.4 depicts an example pseudocode listing of an *Application* frame.

TYPICAL OBJECT MANAGER

```
Include AT_OM.h

/* Define a class that inherits from OMObject class     */
/* eg.  TOP.  This class should contain all the CMIS    */
/* primitive handler functions that Object Manager      */
/* is expected to receive.                              */

class Top : OMObject
  {
  the Constructor
    {
    /* Besides doing whatever is needed to set up       */
    /* your Top object this function must also          */
    /* create the OM object supplying the name of       */
    /* this Object Manager eg.  RTOM                    */

    CreateOM (SomiError, Context Name)

    /* This will attach your Object Manager to          */
    /* the SOMI software bus.                           */

    /* Additionally Top must also register all          */
    /* the classes that it is able to create            */
    /* instances of.  (The class names must exist       */
    /* on the naming tree before instances of           */
    /* them be created.                                 */
    }

  CreateInd (...)
    {
    /* Do whatever you want in response to a CMIS       */
    /* create call.  Typically this will be to          */
    /* create other objects.                            */
    }
  Other CMIS primitive call handlers.
  };
Other classes to be used.


Start of main program
main ( )
  {
  /* Create an instance of your Top object. eg.  Top   */
  while ( True )  pause();
  }
```

Figure C.3: Pseudocode Example of an Object-manager Frame

TYPICAL APPLICATION

```
Include AT_Somi.h

/* Declare the Somi Object ( Global )                    */
/* Declare confirm handlers for each type of             */
/* request that this application is going to use.        */

void CreateCnfHandler( ... )
  {
  Display Returned Message.
  }

void GetCnfHandler ( ... )
  {
  Display returned Message.
  }


Start of main program
main ( )
  {
  /* Attach to SOMI Bus, and tell SOMI about your       */
  /* handlers.  Prepare variables for the CMIS          */
  /* primitives ( SOMI Variable argument lists )        */
  /* eg.  for a "create" the variable argument list     */
  /* contains:                                          */
  /*                                                    */
  /*    MO_CLASS                                        */
  /*    MO_INSTANCE                                     */
  /*    Attribute list.                                 */

  /* Do the Create Request call.                        */
  while ( True ) pause();
  }
```

Figure C.4: Pseudocode Example of an Application Process Frame

APPENDIX D: MODELLING AND IMPLEMENTATION OF THE RTOMS SUBSYSTEM

This Appendix contains a few examples of modelling and class definitions of the RTOMS subsystem. These have been included in the Appendix of this study to serve as an example of the object-oriented analysis, design and class definition process that was undertaken for the project. The examples illustrated are real but have been appropriately abbreviated for the sake of clarity.

Figure D.1 depicts the Class Diagram of the Indication, Equipment, LEG, LEGset model of the RTOMS subsystem. (Refer to the TNM domain descriptions in Chapters 1 and 5 for a review of the TNM domain and associated hierarchy model). Note how the Class Diagram illustrates the static abstractions of the RTOMS subsystem.

Figure D.2 depicts the Object Diagram associated with the Class Diagram. Note how the Object Diagram illustrates the dynamic mechanisms of the object abstractions, such as the 'contains' relationships and the message passing between the objects.

Figures D.3, D.4 and D.5 are listings of the associated C++ header files for the LEG, Equipment and Indication classes.

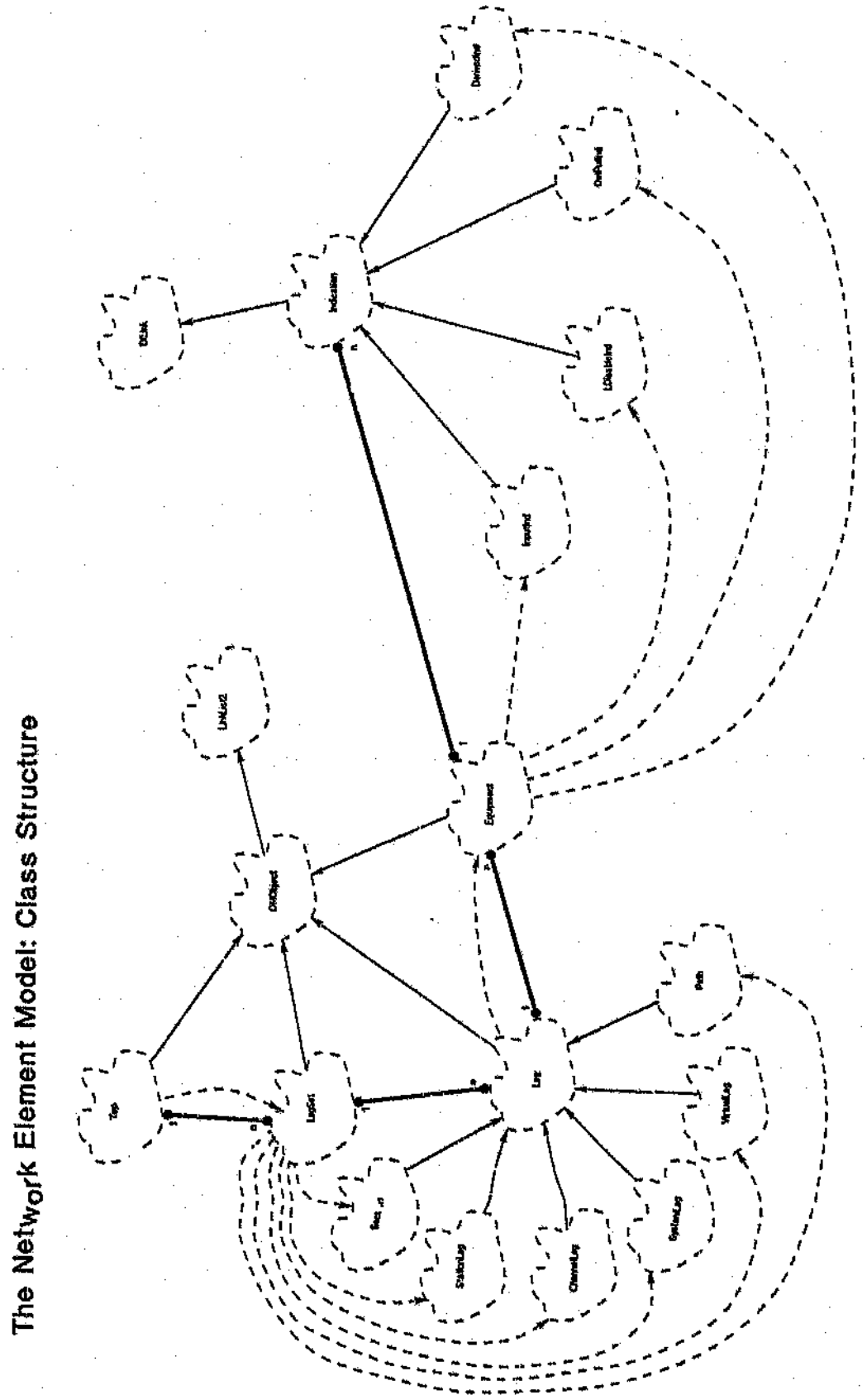


Figure D.1: Example RTOMS Class Diagram

The Network Element Model: Object Structure

Object Model : Expresses the 'contains' relationship of the objects.

Object Model : Expresses the message passing between objects

Figure D.2: Example RTOMS Object Diagram

Figure D.3: C++ Header File for the LEG Class Definition

```
Printed by revision 3.4 of iprt.c
File Name : Leg.h                                   Date : Wed Jan 27 11:35:31 1993   Page :  1

 1  /* This line is for identification by the 'what' command: */
 2  /* @(#)Leg.h    3.2 93/01/26 09:28:19 Aslan \v  RTOM  (C) BSW-Data (Pty) Ltd. */
 3  /*****************************************************************/
 4  /*                                                               */
 5  /*  TITLE BLOCK:                                                 */
 6  /*  ============                                                 */
 7  /*                                                               */
 8  /*  Name         : Leg.h                                         */
 9  /*                                                               */
10  /*  I.D. Number : ASLAN      Rev: 0.00     Rel: 0.00             */
11  /*                                                               */
12  /*    Software Developed By : BSW-DATA                           */
13  /*                            Private Bag X35                    */
14  /*                            Halfway House                      */
15  /*                            1685                               */
16  /*                                                               */
17  /*  Function     : Definition file for the Leg class             */
18  /*                                                               */
19  /*                                                               */
20  /*  History      :                                               */
21  /*                                                               */
22  /*          Event              Date       Author      Approved   */
23  /*                                                               */
24  /*    Specification          92/03/12       PPS          XXX     */
25  /*    Design                 92/03/12       PPS          XXX     */
26  /*    Implementation         92/03/12       PPS          XXX     */
27  /*    Audit                  XX/XX/XX                    XXX     */
28  /*                                                               */
29  /*                                                               */
30  /*  Revisions:  Revision     Date       Author      Approved     */
31  /*                0.00     XX/XX/XX      XXX          XXX       */
32  /*                                                               */
33  /*  Revision Description:                                        */
34  /*  =====================                                        */
35  /*    0.00 - None                                                */
36  /*                                                               */
37  /*                                                               */
38  /*  Functional Specification:                                    */
39  /*  =========================                                    */
40  /*                                                               */
41  /*  Class definition file for Leg                                */
42  /*                                                               */
43  /*****************************************************************/
44
45  #ifndef LEG_INCLUDED
46  #define LEG_INCLUDED
47
48  #include "OH.h"
49  #include "DocketLink.h"
50  #include "OutLink.h"
51  #include "LegSet.h"
52  #include "ExecControl.h"
53
54  /*****************************************************************/
55  /*                                                               */
56  /*  Class Name  : Leg                                            */
57  /*                                                               */
58  /*  Description :                                                */
59  /*  Leg is an acronym for Logical Equipment Group. Its function is to*/
60  /*  contain all the equipment that is logically associated in some  */
61  /*  fashion. For instance, a transmission system consists of various */
62  /*  equipment units linked together via a transmission medium.       */
63  /*                                                               */
64  /*****************************************************************/
65
66  class Leg : public OHObject {
67
68      private:
69
70  /*      Indicates position of attribute in data block            */
71
72          typedef enum {
73              LEG_status = 0,
74              LEG_type,
75              LEG_description,
76              LEG_install_date,
77              LEG_service_date,
78              LEG_service_level,
79              LEG_op_type,
80              LEG_allocation,
81              LEG_ai_supp_flag,
82              LEG_ar_supp_flag,
83              LEG_location_sequence
84              } LEGattrs;
85
86        DocketLink    *first_entry;      /* Pointers to dockets      */
87        DocketLink    *last_entry;
88        ExecControl   *exec_control;     /* Pointer to control object*/
89
90      protected:
91
92  /*          Actions common to all Leg types.                     */
93
94              typedef enum {
95                  Leg_Affected = 0,
96                  Leg_Reconcile,
97                  Leg_Control,
98                  Leg_ExecEnquiry,
99                  Leg_ExecDisable
100             } LegActions;
101
102     public:
103
104
105       OutLink       *first_affects;    /* Pointers to affects links*/
106       OutLink       *last_affects;
107       OutLink       *first_affectedby; /* For affected by links    */
108       OutLink       *last_affectedby;
109       OutLink       *next_entry;
110       Logical       *ar_supp_flag;     /* Alarm suppression flags  */
111       Logical       *ai_supp_flag;
112
113 /*****************************************************************/
114 /*                                                               */
115 /*  Name        : Leg                                            */
116 /*                                                               */
117 /*  Description :                                                */
118 /*  The constructor. Initialises the pointers and flags.         */
119 /*                                                               */
120 /*****************************************************************/
```

```
121     Leg();
122
123 /*****************************************************************/
124 /*                                                               */
125 /* Name        : ~Leg                                            */
126 /*                                                               */
127 /* Description :                                                 */
128 /* The destructor. OMObject will delete the data block.          */
129 /*                                                               */
130 /*****************************************************************/
131     ~Leg();
132
133 /*****************************************************************/
134 /*                                                               */
135 /* Name        : AddCommonAttr                                   */
136 /*                                                               */
137 /* Description :                                                 */
138 /* Invoked by any of the classes inheriting from this class to add */
139 /* all attributes common to all the sub-classes.                 */
140 /*                                                               */
141 /*****************************************************************/
142     void AddCommonAttr();
143
144 /*****************************************************************/
145 /*                                                               */
146 /* Name        : GetId                                           */
147 /*                                                               */
148 /* Description :                                                 */
149 /* This function will return the pointer to the SomiValue used as */
150 /* the ID of this object.                                        */
151 /*                                                               */
152 /*****************************************************************/
153 SomiValue *GetId ( void );              /* Pointer to SomiValue */
154
155 /*****************************************************************/
156 /*                                                               */
157 /* Name        : SetAlarm                                        */
158 /*                                                               */
159 /* Description :                                                 */
160 /* Pass the alarm details to an accepting docket. The docket will */
161 /* returns false if alarm is not accepted.                       */
162 /*                                                               */
163 /*****************************************************************/
164     void SetAlarm (
165     OMObject    *equip,             /* Pointer to equipment     */
166     DlLink      *ind,               /* Pointer to indication    */
167     char        *category,          /* Pointer to category      */
168     char        *criteria,          /* Criteria value           */
169     short       urgency,            /* Urgency of the alarm     */
170     long        severity,           /* Saverity of the alarm    */
171     long        set_time,           /* time of new alarm        */
172     Logical     affected,           /* Network integrity flag   */
173     char        *eqt_id_val,        /* Equipment id pointer     */
174     char        *ind_id_val,        /* Indication id pointer    */
175     Logical     supp_al,            /* Alarm suppression        */
176     Logical     supp_ar);           /* Archive suppression      */
177
178 /*****************************************************************/
179 /*                                                               */
180 /* Name        : ClearAlarm                                      */
181 /*                                                               */
182 /* Description :                                                 */
183 /* Pass the alarm details to an accepting docket. The docket will */
184 /* returns false if the alarm is not accepted.                   */
185 /*                                                               */
186 /*****************************************************************/
187     void ClearAlarm (
188     OMObject    *eqt_ptr,           /* Pointer to equipment     */
189     DlLink      *ind_ptr,           /* Pointer to equipment     */
190     char        *category,          /* Pointer to category      */
191     char        *criteria,          /* Criteria value           */
192     long        set_time,           /* Time of last state change*/
193     long        clear_time,         /* time of alarm clear      */
194     Logical     affected,           /* Network integrity flag   */
195     char        *eqt_id_val,        /* Equipment id pointer     */
196     char        *ind_id_val,        /* Indication id pointer    */
197     Logical     supp_al,            /* Alarm suppression        */
198     Logical     supp_ar);           /* Archive suppression      */
199
200 /*****************************************************************/
201 /*                                                               */
202 /* Name        : ArchiveAlarm                                    */
203 /*                                                               */
204 /* Description :                                                 */
205 /* Sends the archive record to BUCKT.                            */
206 /* Attribute to send                  Where from                 */
207 /*     leg_id                         Docket                     */
208 /*     eqt_id                         Equipment                  */
209 /*     ind_id                         Indication                 */
210 /*     criteria                       Alarm                      */
211 /*     set_time                       Alarm ( last_set_time )    */
212 /*     dref                           Docket                     */
213 /*     det_year  - calculated                                    */
214 /*     det_month - calculated                                    */
215 /*     det_day   - calculated                                    */
216 /*     clear_time                     Alarm                      */
217 /*     locid                          Equipment                  */
218 /*     i_s_id                         Docket                     */
219 /*     val0_num                       Indication                 */
220 /*     val1_num                       Indication                 */
221 /*     val2_num                       Indication                 */
222 /*     val3_num                       Indication                 */
223 /*     val4_num                       Indication                 */
224 /*     val5_num                       Indication                 */
225 /*     cat_id                         Alarm                      */
226 /*     alarm_desc                     Indication                 */
227 /*     flags                          Alarm                      */
228 /*                                                               */
229 /*****************************************************************/
230     void ArchiveAlarm (
231     OMObject    *eqt_ptr,           /* Pointer to the equipment r */
232     DlLink      *ind_ptr,           /* Pointer to the indication  */
233     char        *category,          /* Pointer to category        */
234     char        *criteria,          /* The indication criteria    */
235     long        set_time,           /* Time alarm was set         */
236     long        clear_time,         /* Time alarm was clear       */
237     Logical     supp_ar );          /* Archive suppression flag   */
238
239 /*****************************************************************/
240 /*                                                               */
```

```
241  /*  Name        : ProcessConst                                      */
242  /*                                                                  */
243  /*  Description :                                                   */
244  /*  This function will emulate the CMObject process construction    */
245  /*  But at the same time will set up the condensed record structures*/
246  /*  used by the Leg.                                                */
247  /*                                                                  */
248  /******************************************************************/
249      SomiError ProcessConst(
250      CMObject    *father,                /* The father pointer      */
251      ClassEntry  *ce,                    /* The class entry         */
252      SomiValue   *id,                    /* The id                  */
253      SomiValue   *value,                 /* The value               */
254      CMObject    *ref_obj,               /* The action type         */
255      EXMessage   *message,               /* The message block       */
256      RspBlock    *response);             /* The response block      */
257
258  /******************************************************************/
259  /*                                                                  */
260  /*  Name        : ConstructRecord                                   */
261  /*                                                                  */
262  /*  Description :                                                   */
263  /*  This function will process the record into a compacted version  */
264  /*  specific to each Leg types.                                     */
265  /*                                                                  */
266  /******************************************************************/
267      Logical ConstructRecord (           /* return true if ok       */
268      SomiError   *error,                 /* Return error for failed */
269      SomiValue   *val_ptr,               /* Pointer to value        */
270      void        *value,                 /* Pointer to a value      */
271      int         entry);                 /* The attribute number.   */
272
273  /******************************************************************/
274  /*                                                                  */
275  /*  Name        : ProcessGet                                        */
276  /*                                                                  */
277  /*  Description :                                                   */
278  /*  This function replaces the ProcessGetInd, Leg does some         */
279  /*  terrible things to optimise it memory useage.                   */
280  /*                                                                  */
281  /******************************************************************/
282      SomiError ProcessGet(
283      EXMessage   *message,               /* The message block       */
284      RspBlock    *response,              /* The response block      */
285      SomiValue   *a_id_list);            /* The attribute id list   */
286
287  /******************************************************************/
288  /*                                                                  */
289  /*  Name        : BuildSomi                                         */
290  /*                                                                  */
291  /*  Description :                                                   */
292  /*  Construct the SomiValue from the data block. NOTE: The record   */
293  /*  entry has been specially treated in this case. Remember to free */
294  /*  up the space used by using FreeCopyValue                        */
295  /*                                                                  */
296  /******************************************************************/
297      void BuildSomi(
298      SomiValue   *somi_value,            /* The destination         */
299      int         index);                 /* Attribute index         */
300
301  /******************************************************************/
302  /*                                                                  */
303  /*  Name        : GetRecord                                         */
304  /*                                                                  */
305  /*  Description :                                                   */
306  /*  Get the compressed record into SomiValue construct.             */
307  /*                                                                  */
308  /******************************************************************/
309      Logical GetRecord (                 /* return true if ok       */
310      void        *value,                 /* Pointer to a value      */
311      SomiValue   *val_ptr,               /* Pointer to value        */
312      int         entry);                 /* The attribute number.   */
313
314  /******************************************************************/
315  /*                                                                  */
316  /*  Name        : ProcessDynamicLinks                               */
317  /*                                                                  */
318  /*  Description :                                                   */
319  /*  This function will send its own class and full distinguished    */
320  /*  name to be processed for dynamic link updates and will then     */
321  /*  request all its children to do the same.                        */
322  /*                                                                  */
323  /******************************************************************/
324  void ProcessDynamicLinks(DLMUpdateType update_type);
325
326
327  /******************************************************************/
328  /*                                                                  */
329  /*  Name        : GetLocSequence                                    */
330  /*                                                                  */
331  /*  Description :                                                   */
332  /*  This function will compare the loc_id supplied as input with    */
333  /*  the location sequence record and it will return the position    */
334  /*  as well as the longitude and latitude of the location. If the   */
335  /*  location given is not supplied then this function will return a */
336  /*  -1.                                                             */
337  /*                                                                  */
338  /******************************************************************/
339  short GetLocSequence(
340      char        *loc_id,                /* The location id to find */
341      float       *longitude,             /* The location longitude  */
342      float       *latitude);             /* The location latitude   */
343
344
345  /******************************************************************/
346  /*                                                                  */
347  /*  Name        : SetWordBoundary                                   */
348  /*                                                                  */
349  /*  Description :                                                   */
350  /*  Set position supplied to a word boundary.                       */
351  /*                                                                  */
352  /******************************************************************/
353  int SetWordBoundary ( int current );
354
355  /******************************************************************/
356  /*                                                                  */
357  /*  Name        : GetAttrList                                       */
358  /*                                                                  */
359  /*  Description :                                                   */
360  /*  This function will extract the entire attribute list as         */
```

Printed by revision 3.4 of lprt.c
File Name : Leg.h   Date : Wed Jan 27 11:35:31 1993   Page : 4

```
/*  stored in the data block.                                          */
/*                                                                     */
/*********************************************************************/
void GetAttrList(
     SomiValue    *attr_list,             /* The attribute list       */
     SomiValue    *obj_class,             /* The class list           */
     SomiValue    *obj_instance,          /* The returned instance    */
     long         access);                /* Access permissions       */

/*********************************************************************/
/*                                                                     */
/*  Name        : AddAffectsLinks                                      */
/*                                                                     */
/*  Description :                                                      */
/*  Add Affects links to the list.                                     */
/*                                                                     */
/*********************************************************************/
void AddAffectsLinks(
     SomiError    *error,                 /* Error return value       */
     SomiValue    *id_list,               /* List of attr ids.        */
     SomiValue    *value_list,            /* List of id values        */
     RXMessage    *message,               /* Recieved message         */
     RspBlock     *responce);             /* The response block       */

/*********************************************************************/
/*                                                                     */
/*  Name        : AddAffectedByLinks                                   */
/*                                                                     */
/*  Description :                                                      */
/*  Added AffectedBy links to the list. If the link sequence number   */
/*  is already in use then insert the entry in front of the           */
/*  duplicate and renumber all entries from the duplicate onward.     */
/*                                                                     */
/*********************************************************************/
void AddAffectedByLinks(
     SomiError    *error,                 /* Error return value       */
     SomiValue    *id_list,               /* List of attr ids.        */
     SomiValue    *value_list,            /* List of id values        */
     RXMessage    *message,               /* Recieved message         */
     RspBlock     *response);             /* The response block       */

/*********************************************************************/
/*                                                                     */
/*  Name        : DeleteAffectsLinks                                   */
/*                                                                     */
/*  Description :                                                      */
/*  Delete Affects links from the list.                                */
/*                                                                     */
/*********************************************************************/
void DeleteAffectsLinks(
     SomiError    *error,                 /* Error return value       */
     SomiValue    *id_list,               /* List of attr ids.        */
     SomiValue    *value_list );          /* List of id values        */

/*********************************************************************/
/*                                                                     */
/*  Name        : DeleteAffectedByLinks                                */
/*                                                                     */
/*  Description :                                                      */
/*  Deletes AffectedBy links from the list.                            */
/*                                                                     */
/*********************************************************************/
void DeleteAffectedByLinks(
     SomiError    *error,                 /* Error return value       */
     SomiValue    *id_list,               /* List of attr ids.        */
     SomiValue    *value_list );          /* List of id values        */

/*********************************************************************/
/*                                                                     */
/*  Name        : PropogateStatus                                      */
/*                                                                     */
    Description :                                                      */
/*  Sends the current object status to all Affects entries.            */
/*  The action information needed is as follows:                       */
/*                                                                     */
/*     action_info  ------*                                            */
/*      Record            |                                            */
/*                      Record ------------------------*               */
/*                        |                            |               */
/*                      Record ------------*           |               */
/*                                         |           |               */
/*                                       00000L     "status"           */
/*                                         |           |               */
/*                "Leg" --------------- Record    "source_class"       */
/*                                         |           |               */
/*                "leg_id" ------------ Record    "srce_inst_ids"      */
/*                                         |           |               */
/*                "JSK RTU TP01" ------- Record   "srce_inst_vals"     */
/*                                                                     */
/*  The object instance is constructed so that any object may make   */
/*  use of the affects/affectedby objects. Currently only Leg is      */
/*  capable of using these objects.                                    */
/*                                                                     */
/*********************************************************************/
void PropogateStatus(
     OutLink *specific_link );            /* Pointer to specific link */

/*********************************************************************/
/*                                                                     */
/*  Name        : ProcessStatus                                        */
/*                                                                     */
/*  Description :                                                      */
/*  Processes the affected action. This will search through the       */
/*  AffectedBy links until one is found containing the BOI/BOC as     */
/*  supplied. This entry will then have the status value updated      */
/*  The action information needed is as follows;                       */
/*                                                                     */
/*     action_info  ------*                                            */
/*      Record            |                                            */
/*                      Record ------------------------*               */
/*                        |                            |               */
/*                      Record ------------*           |               */
/*                                         |           |               */
/*                                       00000L     "status"           */
/*                                         |           |               */
/*                "Leg" --------------- Record    "source_class"       */
/*                                         |           |               */
/*                "leg_id" ------------ Record    "srce_inst_ids"      */
/*                                         |           |               */
/*                "JSK RTU TP01" ------- Record   "srce_inst_vals"     */
```

```
481  /*                                                                  */
482  /*  The object instance is constructed so that any object may make  */
483  /*  use of the affects/effectedby objects. Currently only Leg is    */
484  /*  capable of using these objects.                                 */
485  /*                                                                  */
486  /*******************************************************************/
487  SomiError ProcessStatus(
488      SomiValue   *action_info );         /* Action processing info   */
489
490
491  /*******************************************************************/
492  /*                                                                  */
493  /*  Name        : BuildAttributeList                                */
494  /*                                                                  */
495  /*  Description :                                                   */
496  /*  Returns attribute list for the Dynamic Links.                   */
497  /*                                                                  */
498  /*******************************************************************/
499  SomiValue *BuildAttributeList(
500      long        access,             /* Access permissions.          */
501      SomiValue   *a_id_list );       /* Attribute id_list            */
502
503  /*******************************************************************/
504  /*                                                                  */
505  /*  Name        : BuildAttributeList                                */
506  /*                                                                  */
507  /*  Description :                                                   */
508  /*  Returns attribute list for the Dynamic Links.                   */
509  /*                                                                  */
510  /*******************************************************************/
511  SomiValue *BuildAttributeList(
512      SomiValue   *class_ptr,         /* MOC pointer                  */
513      SomiValue   *instance_ptr,      /* MOI pointer                  */
514      ClassEntry  *ci_ptr );          /* Class entry pointer.         */
515
516  /*******************************************************************/
517  /*                                                                  */
518  /*  Name        : ProcessControl                                    */
519  /*                                                                  */
520  /*  Description :                                                   */
521  /*  Initiates the control function processing. If there is not      */
522  /*  already a control in progress then this function will extract   */
523  /*  the action arguments. After validating them the function will   */
524  /*  attempt to fetch the procedure for the control function         */
525  /*  specified. If the procedure does not exist or if any of the     */
526  /*  action arguments are not valid then an error will be returned   */
527  /*  to the initiator of the control.                                */
528  /*                                                                  */
529  /*******************************************************************/
530  void ProcessControl(
531      RspBlock    *response,          /* Response block.              */
532      SomiValue   *action_info );     /* Action arguments             */
533
534  /*******************************************************************/
535  /*                                                                  */
536  /*  Name        : ProcessExecEnquiry                                */
537  /*                                                                  */
538  /*  Description :                                                   */
539  /*  Processes the ExecEnquiry received as an action. The function   */
540  /*  will validate the action_info arguments and if all is OK the    */
541  /*  no error will be returned to the requestee. If the specified    */
542  /*  category does not exist then no error will be returned but with */
543  /*  a flag to indicate this.                                        */
544  /*                                                                  */
545  /*******************************************************************/
546  void ProcessExecEnquiry(
547      RXMessage   *message,           /* Receive message              */
548      RspBlock    *response,          /* Message response block       */
549      SomiValue   *action_info );     /* Action info from request     */
550
551  /*******************************************************************/
552  /*                                                                  */
553  /*  Name        : ExecEnquiry                                       */
554  /*                                                                  */
555  /*  Description :                                                   */
556  /*  Forwards the Execute enquiry request through to all equipments  */
557  /*  ( children ) of this object.                                    */
558  /*                                                                  */
559  /*******************************************************************/
560  Logical ExecEnquiry(                /* False = cat_id not found     */
561      SomiError   *error,             /* Error to be returned         */
562      InvokeID    inv_id,             /* InvokeID to be returned      */
563      void        *sub_key_val,       /* Sub key value                */
564      Logical     display,            /* Return indication attributes */
565      char        *cat_id_ptr,        /* Pointer to category to match */
566      char        *qualifier,         /* Pointer to qual to match     */
567      int         state );            /* new state to set point too   */
568
569  /*******************************************************************/
570  /*                                                                  */
571  /*  Name        : GetCtrlProc                                       */
572  /*                                                                  */
573  /*  Description :                                                   */
574  /*  Requests the parent of this object to return the pointer to the */
575  /*  control procedure indicated by the leg type and function name   */
576  /*                                                                  */
577  /*******************************************************************/
578  SomiRecord *GetCtrlProc(
579      char        *function_name );       /* Name of control function */
580
581  /*******************************************************************/
582  /*                                                                  */
583  /*  Name        : SendDisplay                                       */
584  /*                                                                  */
585  /*  Description :                                                   */
586  /*  Invoked by the Leg if the display flag was set true. This       */
587  /*  function will only be called if an indication was found to be   */
588  /*  valid.                                                          */
589  /*                                                                  */
590  /*******************************************************************/
591  void SendDisplay(
592      InvokeID    inv_id,                 /* The invoke id to use,    */
593      void        *sub_key_val,           /* Sub key value            */
594      char        *eqt_id_ptr,            /* Equipment id pointer     */
595      char        *ind_id_ptr,            /* Indication id pointer    */
596      char        *cat_id,                /* Pointer to category      */
597      char        *alarm_desc,            /* Pointer to Alarm desc    */
598      short       *val_ids,               /* VAL id list.             */
599      SomiError   error );                /* Error return if any.     */
600
```

```
601  /*****************************************************************/
602  /*                                                               */
603  /*  Name        : EnqResponse                                    */
604  /*                                                               */
605  /*  Description :                                                */
606  /*  Response to enquiry. It will be forwarded to the ExecControl */
607  /*  valid.                                                       */
608  /*                                                               */
609  /*****************************************************************/
610  void EnqResponse(
611      InvokeID    inv_id,             /* The invoke id to use.    */
612      void        *sub_key_val,       /* Sub key value            */
613      char        *eqt_id_ptr,        /* Pointer to equipment id  */
614      char        *ind_id_ptr,        /* Pointer to indication id */
615      Logical     state_value,        /* Current state of point   */
616      SomiError   error );            /* Error return if any.     */
617
618  /*****************************************************************/
619  /*                                                               */
620  /*  Name        : ProcessDisables                                */
621  /*                                                               */
622  /*  Description :                                                */
623  /*  This function will validate the incoming message. If the opid*/
624  /*  is valid then this function will extract the filter parameters*/
625  /*  and validate them. If the parameters are valid then all the  */
626  /*  children will be invoked to perform the disable request.     */
627  /*                                                               */
628  /*****************************************************************/
629  void ProcessDisables (
630      RXMessage   *message,           /* The receive message      */
631      RspBlock    *response,          /* The response block       */
632      SomiValue   *action_info );     /* Action info.             */
633
634  /*****************************************************************/
635  /*                                                               */
636  /*  Name        : ControlCompiete                                */
637  /*                                                               */
638  /*  Description :                                                */
639  /*  Cleans up at the End of a Control completion. It is c'ted by */
640  /*  ExecControl.                                                 */
641  /*                                                               */
642  /*****************************************************************/
643  void ControlComplete( void );
644
645  /*****************************************************************/
646  /*                                                               */
647  /*  Name        : SetDisable                                     */
648  /*                                                               */
649  /*  Description :                                                */
650  /*  Loops through all equipment to find an entry with a matching */
651  /*  equipment id. The equipments SetDisable will be invoked and if*/
652  /*  the location matches the given loc_id then the indications will*/
653  /*  will be invoked to set the location disable alarm.           */
654  /*                                                               */
655  /*****************************************************************/
656  void SetDisable(
657      int         function,           /* Type of disable function */
658      char        *locid_ptr,         /* Pointer to a location.   */
659      int         type );             /* Type of LD               */
660
661  /*****************************************************************/
662  /*                                                               */
663  /*  Name        : ClearDisable                                   */
664  /*                                                               */
665  /*  Description :                                                */
666  /*  Loops through all equipment to find an entry with a matching */
667  /*  equipment id. The equipments ClearDisable will be invoked and if*/
668  /*  the location matches the given loc_id then the indications will*/
669  /*  will be invoked to clear the location disable alarm.         */
670  /*                                                               */
671  /*****************************************************************/
672  void ClearDisable(
673      int         function,           /* Type of disable function */
674      char        *locid_ptr,         /* Pointer to a location.   */
675      int         type );             /* Type of LD               */
676
677
678  );  /* Leg */
679  #endif
```

Figure D.4: C++ Header File for the EQUIPMENT Class Definition


```
Printed by revision 3.4 of iprt.c
File Name : Equipment.h                                   Date : Wed Jan 27 08:06:08 1993   Page :  1

/* This line is for identification by the 'what' command! */
/* @(#)Equipment.h  2.8 92/11/09 09:58:49 Aslan Av (RTOM) (C) BSU-Data (Pty) Ltd.
*/
/*****************************************************************************/
/*                                                                           */
/*  TITLE BLOCK:                                                             */
/*  ============                                                             */
/*                                                                           */
/*  Name        : Equipment.h                                                */
/*                                                                           */
/*  I.D. Number : ASLAN     Rev: 0.00     Rel: 0.00                          */
/*                                                                           */
/*    Software Developed by : BSU-DATA                                       */
/*                            Private Bag X33                                */
/*                            Halfway House                                  */
/*                            1685                                           */
/*                                                                           */
/*  Function    : Definition file for the Equipment class.                   */
/*                                                                           */
/*                                                                           */
/*  History     :                                                            */
/*                                                                           */
/*          Event              Date       Author       Approved             */
/*                                                                           */
/*    Specification         92/04/21       PPS            XXX               */
/*    Design                92/04/21       PPS            XXX               */
/*    Implementation        92/04/21       PPS            XXX               */
/*    Audit                 XX/XX/XX                      XXX               */
/*                                                                           */
/*                                                                           */
/*  Revisions:  Revision     Date       Author       Approved                */
/*                 0.00     XX/XX/XX      XXX            XXX                */
/*                 1.00     92/12/14      PSS            AB                 */
/*                                                                           */
/*  Revision Description:                                                    */
/*  =====================                                                    */
/*    0.00 - None                                                            */
/*    1.00 - Model Charger Indications made into para-CMIS objects.          */
/*                                                                           */
/*                                                                           */
/*  Functional Specification:                                                */
/*  =========================                                                */
/*                                                                           */
/*  This is the definition file for the Equipment class                      */
/*                                                                           */
/*****************************************************************************/

#if !defined (EQUIPMENT_H)
#define EQUIPMENT_H

#include "CM.h"
#include "DlLink.h"                     /* Indication link list          */

/*****  ************************************************************/
/* Attribute numbers - last number is always the number of          */
/* attributes contained in objects of this class                    */
/*******************************************************************/
typedef enum {
    EquipmentAtt_status = 0,
    EquipmentAtt_description,
    EquipmentAtt_frequency,
    EquipmentAtt_sequence_no,
    EquipmentAtt_loc_sequence,
    EquipmentAtt_loc_s_long,
    EquipmentAtt_loc_s_lat,
    EquipmentAtt_code,
    EquipmentAtt_locId,
} EquipmentAtt_No;

typedef enum {
    Equipment_reconcile = 0,
} EquipmentActions;

/*****************************************  *****************/
/*                                                          */
/*  Class       : Equipment                                 */
/*                                                          */
/*  Description :                                           */
/*  This cl   contains CMIS information specific to the actual */
/*  Network   ament. This class also acts as a container for the */
/*  objects ( indications ) representing the outputs generated by */
/*  the Network Element ( Equipment ). These outputs are monitored by */
/*  Surveillance Equipment Model ( SEM ) and the UMI. Any change of */
/*  state generated by the NE is detected by the UMI, passed to the */
/*  SEM and thence to the indications contained by this class.  */
/*  When the indication processes the state change it will initiate */
/*  the generation of Dockets/Alarms via this class.            */
/*  NOTE:                                                       */
/*  1.  The status of this object represents the cumulative status */
/*      of all its indications.                                 */
/*  2.  Because of the memory cost implications all indications are */
/*      implemented as PARA-CMIS objects. IE. They do not make use */
/*      of standard CMObject constructs and everything destined for */
/*      indications must pass through this object.              */
/*                                                              */
/****************************************************************/
class Equipment : public CMObject {

    private:

        static Logical first_instance;
        static ClassBluePrint *blue_print;

    protected:

        DlLink  *first_ind;         /* Pointer to first indication */
        DlLink  *last_ind;          /* Pointer to last indication  */

    public:

/*****************************************************************/
/*                                                               */
/*  Name        : Equipment                                      */
/*                                                               */
/*  Description : The constructor for the Equipment class        */
/*                                                               */
/*****************************************************************/
Equipment (
    ScalError   *error,             /* Error returned to parent  */
```

```
120      CMObject    *father,          /* Pointer to parent          */
121      ClassEntry  *ci_entry,        /* Pointer to class entry     */
122      SomiValue   *id,              /* Instance id id             */
123      SomiValue   *value,           /* Instance id value          */
124      CMObject    *ref_obj,         /* A refence object           */
125      RXMessage   *message,         /* The received message       */
126      RspBlock    *response );      /* A response block           */
127
128
129  /*****************************************************************/
130  /*                                                               */
131  /*  Name        : ~Equipment()                                   */
132  /*                                                               */
133  /*  Description : The desctructor.                               */
134  /*                                                               */
135  /*****************************************************************/
136  ~Equipment ();
137
138  /*****************************************************************/
139  /*  Name        : CreateInd                                      */
140  /*  Description :                                                */
141  /*  Equipment creates instances of the following classes:        */
142  /*      InputInd.                                                */
143  /*      OutputInd.                                               */
144  /*      LOInd,                                                   */
145  /*      DerivedInd.                                              */
146  /*                                                               */
147  /*****************************************************************/
148  void CreateInd(
149      RXMessage   *message,             /* The received message   */
150      RspBlock    *response,            /* Response block         */
151      ClassEntry  *ci_entry,            /* Related class entry    */
152      SomiValue   *id,                  /* New object id id       */
153      SomiValue   *value,               /* New object id value    */
154      CMObject    *ref_object );        /* A reference object     */
155
156  /*****************************************************************/
157  /*                                                               */
158  /*  Name        : DeleteInd                                      */
159  /*                                                               */
160  /*  Description : Deletes the named objects. This will allways   */
161  /*  delete the children. Remember that the childs children must be */
162  /*  cared for as well.                                           */
163  /*                                                               */
164  /*****************************************************************/
165  void DeleteInd( RXMessage *message,       /* The Receive message */
166                  RspBlock *response,       /* The response block  */
167                  ClassEntry *ci_entry,     /* The class entry ptr */
168                  CMObject *del_object);    /* The object to delete */
169
170  /*****************************************************************/
171  /*                                                               */
172  /*  Name        : GetInd                                         */
173  /*                                                               */
174  /*  Description : Retrieves all t.       utes specified. If the  */
175  /*  attribute list contains an ent.      does not exist or is not */
176  /*  visible then an error must be ret.   .d.                     */
177  /*                                                               */
178  /*****************************************************************/
179  void GetInd(    RXMessage *message,       /* The receive message */
180                  RspBlock *response,       /* The response block  */
181                  SomiValue
182                          *attr_id_list);   /* Attr to be returned */
183
184  /*****************************************************************/
185  /*                                                               */
186  /*  Name        : SetInd                                         */
187  /*                                                               */
188  /*  Description : Sets the specified attributes to the supplied  */
189  /*  values. The attributes must first be validated before any    */
190  /*  changes are made and an error must be returned if any of the */
191  /*  AVA entries are incorrect.                                   */
192  /*                                                               */
193  /*****************************************************************/
194  void SetInd(    RXMessage *message,       /* The receive message */
195                  RspBlock *response,       /* The response block  */
196                  SomiValue
197                          *attr_mod_list);  /* Attr to be modified */
198
199  /*****************************************************************/
200  /*                                                               */
201  /*  Name        : ActionInd                                      */
202  /*                                                               */
203  /*  Description : Performs some sort of action.                  */
204  /*                                                               */
205  /*****************************************************************/
206  void ActionInd( RXMessage *message,       /* The receive message */
207                  RspBlock *response,       /* The response block  */
208                  SomiValue *action_type,   /* Type of action      */
209                  SomiValue *action_info);  /* Action information  */
210
211  /*****************************************************************/
212  /*                                                               */
213  /*  Name        : CreateCnf                                      */
214  /*                                                               */
215  /*  Description : This routine is called if this object had      */
216  /*  requested that an object be created via SOMI                 */
217  /*                                                               */
218  /*****************************************************************/
219  void CreateCnf( RXMessage *rxm);          /* The receive message */
220
221  /*****************************************************************/
222  /*                                                               */
223  /*  Name        : DeleteCnf                                      */
224  /*                                                               */
225  /*  Description : This routine is called if this object had      */
226  /*  requested that an object be deleted via SOMI                 */
227  /*                                                               */
228  /*****************************************************************/
229  void DeleteCnf( RXMessage *rxm);          /* The receive message */
230
231  /*****************************************************************/
232  /*                                                               */
233  /*  Name        : GetCnf                                         */
234  /*                                                               */
235  /*  Description : This routine is called if this object had      */
236  /*  requested a get via SOMI.                                    */
237  /*                                                               */
238  /*****************************************************************/
239  void GetCnf(    RXMessage *rxm);          /* The receive message */
```

```
240
241  /*****************************************************************/
242  /*                                                               */
243  /*  Name         : SetCnf                                        */
244  /*                                                               */
245  /*  Description : This routine is called if this object had      */
246  /*  requested a set via SOMI.                                    */
247  /*                                                               */
248  /*****************************************************************/
249  void SetCnf(    RXMessage *rxm);          /* The receive message  */
250
251  /*****************************************************************/
252  /*                                                               */
253  /*  Name         : ActionCnf                                     */
254  /*                                                               */
255  /*  Description : This routine is called if this object had      */
256  /*  requested that an action be perforzed via SOMI               */
257  /*                                                               */
258  /*                                                               */
259  /*                                                               */
260  /*                                                               */
261  /*****************************************************************/
262  void ActionCnf( RXMessage *rxm);          /* The receive message  */
263
264  /*****************************************************************/
265  /*                                                               */
266  /*  Name         : EventReportCnf                                */
267  /*                                                               */
268  /*  Description : This routine is called if this object had      */
269  /*  Sent out an EventReport in confirmed mode.                   */
270  /*                                                               */
271  /*****************************************************************/
272  void EventReportCnf( RXMessage *rxm);     /* The receive message  */
273
274  /*****************************************************************/
275  /*                                                               */
276  /*  Name         : SetAlarm                                      */
277  /*                                                               */
278  /*  Description :                                                */
279  /*  This function will send a set alarm to the leg for inclusion in */
280  /*  a docket.                                                    */
281  /*                                                               */
282  /*****************************************************************/
283  void SetAlarm (
284      DlLink      *ind,                 /* Pointer to indication   */
285      char        *category,            /* Pointer to category     */
286      char        *criteria,            /* Criteria of alarm       */
287      short       urgency,              /* Urgency value           */
288      long        severity,             /* Severity of alarm       */
289      long        set_time,             /* Set time of the alarm   */
290      Logical     affected,             /* Network integrity flag  */
291      char        *ind_id_val,          /* Indication id value     */
292      Logical     supp_al,              /* Alarm suppression       */
293      Logical     supp_ar);             /* Archive suppression     */
294
295  /*****************************************************************/
296  /*                                                               */
297  /*  Name         : ClearAlarm                                    */
298  /*                                                               */
299  /*  Description :                                                */
300  /*  This function will send a clear alarm to the leg for passing */
301  /*  on to a docket.                                              */
302  /*                                                               */
303  /*****************************************************************/
304  void ClearAlarm (
305      DlLink      *ind_ptr,             /* Pointer to indication   */
306      char        *category,            /* Pointer to category     */
307      char        *criteria,            /* Criteria of alarm       */
308      long        set_time,             /* Time of previous change */
309      long        clear_time,           /* clear_time of the alarm */
310      Logical     affected,             /* Network integrity flag  */
311      char        *ind_id_val,          /* Indication id           */
312      Logical     supp_al,              /* Alarm suppression       */
313      Logical     supp_ar );            /* Archive suppression     */
314
315  /*****************************************************************/
316  /*                                                               */
317  /*  Name         : GetId                                         */
318  /*                                                               */
319  /*  Description :                                                */
320  /*  This function will return the pointer to the SomiValue used as */
321  /*  the ID of this object.                                       */
322  /*                                                               */
323  /*****************************************************************/
324  SomiValue *GetId ( void );                /* Pointer to SomiValue */
325
326  /*****************************************************************/
327  /*                                                               */
328  /*  Name         : ProcessDynamicLinks                           */
329  /*                                                               */
330  /*  Description :                                                */
331  /*  This function will send its own class and full distinguished */
332  /*  name to be processed for dynamic link updates and will then  */
333  /*  request all its children to do the same.                     */
334  /*                                                               */
335  /*****************************************************************/
336  void ProcessDynamicLinks(DLNUpdateType update_type);
337
338  /*****************************************************************/
339  /*                                                               */
340  /*  Name         : GetLocSequence                                */
341  /*                                                               */
342  /*  Description :                                                */
343  /*  Get the location sequence number, the equipment sequence number */
344  /*  the locations latitude and longitude as well as the location */
345  /*  id.                                                          */
346  /*                                                               */
347  /*****************************************************************/
348  short GetLocSequence (              /* Return location sequence num */
349      char    *loc_id,                /* Pointer for location id  */
350      float   *longitude,             /* Pointer to longitude     */
351      float   *latitude,              /*       and latitude       */
352      short   *equip_seq );           /* Equipment sequence number */
353
354  /*****************************************************************/
355  /*                                                               */
356  /*  Name         : ExecEnquiry                                   */
357  /*                                                               */
358  /*  Description :                                                */
359  /*  Forwards the Execute enquiry request through to all equipments */
```

```
360  /*  ( children ) of this object.                                    */
361  /*                                                                   */
362  /*******************************************************************/
363  Logical ExecEnquiry(                /* False = cat_id not found     */
364      SomiError   *error,             /* Error to be returned         */
365      InvokeID    inv_id,             /* Invoke ID used as key        */
366      void        *sub_key_val,       /* Sub key value                */
367      Logical     display,            /* Return indication attributes */
368      char        *cat_id_ptr,        /* Pointer to category to match */
369      char        *qualifier,         /* Pointer to qual to match     */
370      int         state );            /* new state to set point too   */
371
372  /*******************************************************************/
373  /*                                                                   */
374  /*  Name        : SendDisplay                                        */
375  /*                                                                   */
376  /*  Description :                                                    */
377  /*  Invoked by the indication if the display flag was set true.      */
378  /*                                                                   */
379  /*******************************************************************/
380  void SendDisplay(
381      InvokeID    inv_id,             /* The invoke id to use.        */
382      void        *sub_key_val,       /* Sub key value                */
383      char        *ind_id_ptr,        /* Indication id pointer        */
384      char        *cat_id,            /* Pointer to category          */
385      char        *alarm_desc,        /* Pointer to Alarm des.        */
386      short       *mmi_ids,           /* List of MMI ids              */
387      SomiError   error );            /* Error retur. if any.         */
388
389  /*******************************************************************/
390  /*                                                                   */
391  /*  Name        : EngResponse                                        */
392  /*                                                                   */
393  /*  Description :                                                    */
394  /*  Invoked by the Leg if the display flag was set true. This        */
395  /*  function will only be called if an indication was found to be    */
396  /*  valid.                                                           */
397  /*                                                                   */
398  /*******************************************************************/
399  void EngResponse(
400      InvokeID    inv_id,             /* The invoke id to use.        */
401      voiu        *sub_key_val,       /* Sub key value                */
402      char        *ind_id_ptr,        /* Pointer to indication id     */
403      Logical     state_value,        /* Current state of point       */
404      SomiError   error );            /* Error return if any.         */
405
406  /*******************************************************************/
407  /*                                                                   */
408  /*  Name        : AddIndication                                      */
409  /*                                                                   */
410  /*  Description :                                                    */
411  /*  Add an indication to the list provided it doesn't already exist.*/
412  /*                                                                   */
413  /*******************************************************************/
414  SomiError AddIndication(
415      SomiValue   *id_list,           /* List of attr ids.            */
416      SomiValue   *value_list,        /* List of id values            */
417      RXMessage   *message );         /* Recieved message             */
418
419  /*******************************************************************/
420  /*                                                                   */
421  /*  Name        : DeleteIndications                                  */
422  /*                                                                   */
423  /*  Description :                                                    */
424  /*  Delete Indicators from the list.                                 */
425  /*                                                                   */
426  /*******************************************************************/
427  void DeleteInd'cations(
428      SomiError   *error,             /* Error return value           */
429      SomiValue   *id_list,           /* List of attr ids.            */
430      SomiValue   *value_list );      /* List of id values            */
431
432  /*******************************************************************/
433  /*                                                                   */
434  /*  Name        : ProcessDisables                                    */
435  /*                                                                   */
436  /*  Description :                                                    */
437  /*  This function will process the Logical disable requests.         */
438  /*                                                                   */
439  /*******************************************************************/
440  void ProcessDisables(
441      int         function,           /* Type of ID to perform        */
442      Logical     disable_ctrl,       /* Level of disable control     */
443      int         type,               /* Type of disable ( crit )     */
444      char        *locid_mask,        /* Location mask                */
445      char        *catid_mask );      /* Category mask                */
446
447  /*******************************************************************/
448  /*                                                                   */
449  /*  Name        : SetDisable                                         */
450  /*                                                                   */
451  /*  Description :                                                    */
452  /*  If the given locid matches that of equipment then the SetDisable*/
453  /*  function of indication will be invoked to set the correct alarm  */
454  /*  if the correct alarm was found then the function will return     */
455  /*  with true.                                                       */
456  /*                                                                   */
457  /*******************************************************************/
458  Logical SetDisable(
459      int         function,           /* Type of function.            */
460      char        *locid_mask,        /* Location mask                */
461      int         type );             /* Type of disable ( crit )     */
462
463  /*******************************************************************/
464  /*                                                                   */
465  /*  Name        : ClearDisable                                       */
466  /*                                                                   */
467  /*  Description :                                                    */
468  /*  If the given locid matches that of equipment then the            */
469  /*  ClearDisable function of indication will be invoked to clear     */
470  /*  the correct alarm if the correct alarm was found then the        */
471  /*  function will return with true.                                  */
472  /*                                                                   */
473  /*******************************************************************/
474  Logical ClearDisable(
475      int         function,           /* Type of function.            */
476      char        *locid_mask,        /* Location mask                */
477      int         type );             /* Type of disable ( crit )     */
478
479  };  /* Class Equipment */
```

Figure D.5: C++ Header File for the INDICATION Class Definition

```
Printed by revision 3.4 of iprt.c
File Name : Indication.h                    Date : Wed Jan 27 08:09:23 1993          Page :  1

   1  /* This line is for identification by the 'what' command[ */
   2  /* %Z% %M% %I% Aslan Av [RT-MU (C) BSW-Data (Pty) Ltd. */
   3  /*****************************************************************/
   4  /*                                                               */
   5  /*  TITLE BLOCK:                                                 */
   6  /*  ************                                                 */
   7  /*                                                               */
   8  /*  Name        : Indication.h                                   */
   9  /*                                                               */
  10  /*  I.D. Number : ASLAN     Rev: 0.00     Ref: 0.00              */
  11  /*                                                               */
  12  /*    Software Developed By : BSW-DATA                           */
  13  /*                            Private Bag X35                    */
  14  /*                            Halfway House                      */
  15  /*                            1685                               */
  16  /*                                                               */
  17  /*  Function    : Base class for all indications.                */
  18  /*                                                               */
  19  /*                                                               */
  20  /*  History     :                                                */
  21  /*                                                               */
  22  /*          Event              Date      Author      Approved     */
  23  /*                                                               */
  24  /*    Specification        92/12/04      PBB          AB         */
  25  /*    Design               92/12/04      PBB          AB         */
  26  /*    Implementation       92/12/04      PBB          AB         */
  27  /*    Audit                XX/XX/XX                   XXX        */
  28  /*                                                               */
  29  /*                                                               */
  30  /*  Revisions:  Revision     Date      Author      Approved       */
  31  /*                0.00     XX/XX/XX     XXX          XXX         */
  32  /*                                                               */
  33  /*  Revision Description:                                        */
  34  /*  *********************                                        */
  35  /*    0.00 - None                                                */
  36  /*                                                               */
  37  /*                                                               */
  38  /*  Functional Specifications:                                   */
  39  /*  **************************                                   */
  40  /*                                                               */
  41  /*  Contains the class description for the Indication class.     */
  42  /*                                                               */
  43  /*****************************************************************/
  44  #ifndef INDICATION_INCLUDED
  45  #define INDICATION_INCLUDED
  46
  47  #include "OK.h"
  48  #include "RtosTypes.h"
  49  #include "DiLink.h"
  50
  51  /*  Bitmap values for indication bitmap.                         */
  52
  53  #define IND_AR_SUPP       010000000     /* Default settings      */
  54  #define IND_AL_SUPP       004000000     /* Default settings      */
  55  #define IND_CURR_AR_SUPP  002000000     /* Current settings      */
  56  #define IND_CURR_AL_SUPP  001000000     /* Current settings      */
  57  #define IND_AR_DISABLE    000400000     /* Disable in force      */
  58  #define IND_AL_DISABLE    000200000     /* Disable in force      */
  59  #define IND_AR_ENABLE     000100000     /* Enable  in force      */
  60  #define IND_AL_ENABLE     000040000     /* Enable  in force      */
  61  #define IND_INTEGRITY     000020000     /* Integrity problem     */
  62  #define IND_RECONCILE     000010000     /* Reconcile in progress */
  63  #define IND_LEG_AR_DIS    000004000     /* Legs archive disable flag*/
  64  #define IND_LEG_AL_DIS    000002000     /* Legs alarm disable flag */
  65  #define IND_STATE_CHANGE  000001000     /* State change during disab*/
  66  #define IND_LDAR_PER_LOC  000000400     /* Disabled per location   */
  67  #define IND_LDAL_PER_LOC  000000200     /* Disabled per location   */
  68  #define IND_LDAR_PER_IND  000000100     /* Disabled per indication */
  69  #define IND_LDAL_PER_IND  000000040     /* Disabled per indication */
  70  #define IND_STATE         000000001     /* Indication state        */
  71
  72  /*  Some macros to make things clearer                           */
  73
  74  #define FlagIsClear( a ) (0 == ( ind_bitmap & (long) a ) )
  75  #define FlagIsSet( a ) (( ind_bitmap & (long) a ) )
  76  #define SetFlag( a ) (ind_bitmap = ( ind_bitmap | (long) a ) )
  77  #define ClearFlag( a ) (ind_bitmap = ( ind_bitmap & ~(long)a ) )
  78
  79  /*  Link types ( For Wire links )                                */
  80
  81  #define IND_WIRE          0
  82  #define IND_UPLINK        1
  83  #define IND_DOWNLINK      2
  84  #define IND_WALFAULT_WIRE    99
  85
  86  /*****************************************************************/
  87  /*                                                               */
  88  /*  Class Name  : Indication                                     */
  89  /*                                                               */
  90  /*  Description :                                                */
  91  /*  This class contains all the data and functions that are common */
  92  /*  to all the indication classes.                               */
  93  /*  Because of the memory implications this object will be a     */
  94  /*  para-CMIS object including any link details ( Wire etc )    */
  95  /*  The links are contained in a special structure to optimise the */
  96  /*  amount of memory used by these objects. The Input/Output Ind's */
  97  /*  support a Wire and any number of Uplinks. While the Derived  */
  98  /*  supports Uplinks and Downlinks. Each of these types are      */
  99  /*  classified using a number:                                   */
 100  /*  0 --> Wire entry                                             */
 101  /*  1 --> Uplink entry.                                          */
 102  /*  2 --> Downlink entry.                                        */
 103  /*  99 --> Wire to WALFaults                                     */
 104  /*  These entries will be packed into a byte string as follows:  */
 105  /*  Depending on the number of supplied entries the string will be */
 106  /*  malloc'd to the correct size. The string will contain each entry*/
 107  /*  in compressed format. The first byte will indicate the type  */
 108  /*  while the following n bytes will contain the entry depending on */
 109  /*  the type. Thereafter each entry will follow on. The lengths of */
 110  /*  these entry will always be dependent on the entry type.      */
 111  /*                                                               */
 112  /*  Wire:    2 byte type.                                        */
 113  /*           14 bytes containing the full WAL id as short values */
 114  /*  Uplink/Downlink:    2 byte type.                             */
 115  /*           40 bytes containing the destination ID as LegSet(short)*/
 116  /*           Leg(12 char ), Equipment(12 char ) and Indication   */
 117  /*           (8 char) and the input and output attributes as shorts */
 118  /*  Wire to WALFaults:                                           */
 119  /*           2 byte type.                                        */
 120  /*           2 byte number of elements in structure              */
```

Printed by revision 3.4 of lprt.c
File Name : indication.h Date : Wed Jan 27 08:09:23 1993 Page : 1

```
121  /*          n elements of two bytes each.                          */
122  /*                                                                 */
123  /*-----------------------------------------------------------------*/
124  class Indication : public DLLink    {
125      private:
126
127      protected:
128
129          char        ind_id[IND_ID_SIZE];    /* The id of this obj  */
130          long        status;                 /* Object status value */
131          char        ind_type[2];            /* Indication type     */
132          short       urgency;                /* Urgency value       */
133          long        ind_bitmap;             /* Indication bitmap   */
134          char        cat_id[CAT_ID_SIZE];    /* Full category       */
135          char        alarm_label[ALARM_LABEL_SIZE];
136                                              /* Label of alarm      */
137          long        severity;               /* Configured severity */
138          long        last_tr_time;           /* Time of last trans  */
139          struct  {
140              int     n_elmts;                /* Number of entries   */
141              char    *list;                  /* Pointer to list.    */
142          }   links;                          /* Links to other obj's */
143
144      public:
145  /*-----------------------------------------------------------------*/
146  /*                                                                 */
147  /*  Name        : Indication                                       */
148  /*                                                                 */
149  /*  Description :                                                  */
150  /*  The constructor. Initialises the general variables.            */
151  /*                                                                 */
152  /*-----------------------------------------------------------------*/
153  Indication ( void );
154
155  /*-----------------------------------------------------------------*/
156  /*                                                                 */
157  /*  Name        : ~Indication                                      */
158  /*                                                                 */
159  /*  Description :                                                  */
160  /*  This function will clean up all resources used by this class   */
161  /*                                                                 */
162  /*-----------------------------------------------------------------*/
163  ~Indication ( );
164
165  /*-----------------------------------------------------------------*/
166  /*                                                                 */
167  /*  Name        : ExtractCommonAttr                                */
168  /*                                                                 */
169  /*  Description :                                                  */
170  /*  Extract all attributes from the supplied list that are common  */
171  /*  to all indications.                                            */
172  /*                                                                 */
173  /*-----------------------------------------------------------------*/
174  SomiError ExtractCommonAttr(            /* Return Error if failed  */
175      SomiValue   *attrs );               /* Structure to process    */
176
177  /*-----------------------------------------------------------------*/
178  /*                                                                 */
179  /*  Name        : CompressLinks                                    */
180  /*                                                                 */
181  /*  Description :                                                  */
182  /*  Pack and compress the destination ids according to the rules   */
183  /*  indica. d by the class.                                        */
184  /*                                                                 */
185  /*-----------------------------------------------------------------*/
186  SomiError CompressLinks(                /* Return Error if failed  */
187      SomiRecord  *record );              /* The record to compress  */
188
189  /*-----------------------------------------------------------------*/
190  /*                                                                 */
191  /*  Name        : BuildLinks                                       */
192  /*                                                                 */
193  /*  Description :                                                  */
194  /*  Unpack the compressed link into a malloced somivalue structure */
195  /*                                                                 */
196  /*-----------------------------------------------------------------*/
197  SomiValue *BuildLinks(                  /* Return SomiValues       */
198      SomiError   *error );               /* Return error if failed  */
199
200  /*-----------------------------------------------------------------*/
201  /*                                                                 */
202  /*  Name        : CleanUp                                          */
203  /*                                                                 */
204  /*  Description :                                                  */
205  /*  Cleans up all the malloc'd structures created by BuildLink.    */
206  /*  Specific attention needed here because the structure may not   */
207  /*  have been completely filled so the routine must check before   */
208  /*  freeing up any space.                                          */
209  /*                                                                 */
210  /*-----------------------------------------------------------------*/
211  void CleanUp(
212      SomiValue   *record_ptr );          /* Record to clean up      */
213
214  /*-----------------------------------------------------------------*/
215  /*                                                                 */
216  /*  Name        : GetWAL                                           */
217  /*                                                                 */
218  /*  Description :                                                  */
219  /*  Copies the WAL ids into the supplied array.                    */
220  /*                                                                 */
221  /*-----------------------------------------------------------------*/
222  int GetWAL(                             /* Return -1 if unconfigured*/
223      short   *array );                   /* Array to return values in*/
224
225  /*-----------------------------------------------------------------*/
226  /*                                                                 */
227  /*  Name        : GetStatus                                        */
228  /*                                                                 */
229  /*  Description :                                                  */
230  /*  Returns the status word of this object.                        */
231  /*                                                                 */
232  /*-----------------------------------------------------------------*/
233  long GetStatus(                         /* Return status w         */
234      void ) {
235      return ( status );
236  }
237
238  /*-----------------------------------------------------------------*/
239  /*                                                                 */
240  /*  Name        : GetAlarmLabel                                    */
```

```
241  /*                                                                    */
242  /*  Description :                                                     */
243  /*  Returns a pointer to the alarm label.                             */
244  /*                                                                    */
245  /**********************************************************************/
246  char *GetAlarmLabel(                    /* Return pointer to descrip*/
247      void ) {
248      return ( alarm_label );
249  }
250
251  /**********************************************************************/
252  /*                                                                    */
253  /*  Name        : GetId                                               */
254  /*                                                                    */
255  /*  Description :                                                     */
256  /*  Returns a pointer to the indication id.                           */
257  /*                                                                    */
258  /**********************************************************************/
259  char *GetId(                            /* Return id of object      */
260      void ) {
261      return( ind_id );
262  }
263
264  /**********************************************************************/
265  /*                                                                    */
266  /*  Name        : TestEntry                                           */
267  /*                                                                    */
268  /*  Description :                                                     */
269  /*  Return true if the id value pair supplied matches that of this    */
270  /*  entry.                                                            */
271  /*                                                                    */
272  /**********************************************************************/
273  Logical TestEntry (                     /* Return true if match     */
274      SmiValue    *ids,                   /* The id id                */
275      SmiValue    *values );              /* The id value             */
276
277  /**********************************************************************/
278  /*                                                                    */
279  /*  Name        : SetInd                                              */
280  /*                                                                    */
281  /*  Description :                                                     */
282  /*  This function will extract the modifier values and depending on   */
283  /*  the exact attributes the following will be assumed:               */
284  /*  1. If the last_tt_time is supplied then this is assumed to be a   */
285  /*     state change and it will be processed as such.                 */
286  /*  2. Otherwise this entry will be processed as a normal set and     */
287  /*     all attributes will bve extracted and updated.                 */
288  /*                                                                    */
289  /**********************************************************************/
290  SmiError SetInd (                       /* Return error             */
291      RspBlock    *response,              /* Response block           */
292      OMObject    *parent,                /* Pointer to parent        */
293      SmiValue    *attrs );               /* Attribute modifiers      */
294
295  /**********************************************************************/
296  /*                                                                    */
297  /*  Name        : ProcessStateChange                                  */
298  /*                                                                    */
299  /*  Description :                                                     */
300  /*  This function will construct a var_arg structure and send it      */
301  /*  out using the fathers OK pointer.                                 */
302  /*                                                                    */
303  /**********************************************************************/
304  void ProcessStateChange (
305      long    tt_time,                    /* Time of last transition  */
306      Logical new_state,                  /* New state of indication  */
307      char    *crit_label );              /* Criteria label           */
308
309  /**********************************************************************/
310  /*                                                                    */
311  /*  Name        : GetInd                                              */
312  /*                                                                    */
313  /*  Description :                                                     */
314  /*  This function will construct a var_arg structure and send it      */
315  /*  out using the fathers OK pointer.                                 */
316  /*                                                                    */
317  /**********************************************************************/
318  SmiError GetInd (                       /* Return error             */
319      RspBlock    *response,              /* Response block           */
320      OMObject    *father,                /* Pointer to father        */
321      Logical     linked = BoolFalse );   /* Scoped replies           */
322
323  /**********************************************************************/
324  /*                                                                    */
325  /*  Name        : GetPcClassAndFOK                                    */
326  /*                                                                    */
327  /*  Description :                                                     */
328  /*  Builds up a class and instance id list of objects in this         */
329  /*  objects naming tree including this object which happens to be     */
330  /*  a Para-CHIS object.                                               */
331  /*  NOTE: The SmiValue variables classes and instances must be        */
332  /*        pointers to existing SmiValues.                             */
333  /*                                                                    */
334  /**********************************************************************/
335  SmiError GetPcClassAndFOK (             /* Return error if failed   */
336      OMObject    *father,                /* Pointer to the father    */
337      SmiVal      *classes,               /* The class list           */
338      SmiVa       *instances,             /* Instance ids and values  */
339      SmiVals     *my_class,              /* This objects class       */
340      SmiValue    *my_id,                 /* This objects id none     */
341      SmiValue    *my_value );            /* This objects id value    */
342
343  /**********************************************************************/
344  /*                                                                    */
345  /*  Name        : BuildAttributes                                     */
346  /*                                                                    */
347  /*  Description :                                                     */
348  /*  Build up the complete attribute list supplied.                    */
349  /*                                                                    */
350  /**********************************************************************/
351  SmiError BuildAttributes(               /* Return error if problems */
352      SmiValue    *classes,               /* Pointer to class list    */
353      SmiValue    *instances,             /* Pointer to instance list */
354      SmiValue    *attr_list );           /* Pointer to attr list.    */
355
356  /**********************************************************************/
357  /*                                                                    */
358  /*  Name        : TestId                                              */
359  /*                                                                    */
360  /*  Description :                                                     */
```

Printed by revision 3.4 of lprt.c
File Name : Indication.h   Date : Wed Jan 27 08:09:23 1993   Page : 4

```
/*  Validate the supplied SomiValue id against this objects id.      */
/*                                                                   */
/*******************************************************************/
Logical TestId(                        /* Return True if match      */
    SomiValue   *id_val );             /* SomiValue pointer         */

/*******************************************************************/
/*                                                                   */
/*  Name        : ProcessDynamicLinks                                */
/*                                                                   */
/*  Description :                                                    */
/*  This function will send of a DLMCallBackRsp.                     */
/*                                                                   */
/*******************************************************************/
SomiError ProcessDynamicLinks(         /* Return error for fail     */
    OMObject        *father,           /* Pointer to parent         */
    DLMUpdateType   type );            /* Class Id list             */

/*******************************************************************/
/*                                                                   */
/*  Name        : ExecEnquiry                                        */
/*                                                                   */
/*  Description :                                                    */
/*  Validate category against internal criteria and send an enquiry */
/*  to WALS if valid. If the display flag is set then send display  */
/*  Information to the Leg for forwarding to the ExecControl object */
/*                                                                   */
/*******************************************************************/
Logical ExecEnquiry(                   /* False = cat_id not found  */
    SomiError   *error,                /* Error to be returned      */
    InvokeID    inv_id,                /* The invoke id used as key */
    void        *sub_key_val,          /* Sub key value             */
    Logical     display,               /* Return Indication attributes */
    char        *cat_id_ptr,           /* Pointer to category to match */
    int         state );               /* new state to set point too */

/*******************************************************************/
/*                                                                   */
/*  Name        : SendEnquiry                                        */
/*                                                                   */
/*  Description :                                                    */
/*  This will send the enquiry off to the wired WALS ( if any )      */
/*                                                                   */
/*******************************************************************/
InvokeID SendEnquiry(                  /* Return the invoke id      */
    SomiError   *error,                /* Error return value        */
    Logical     *flag,                 /* Indicates wait for resp   */
    int         new_state );           /* New state of crit.        */

/*******************************************************************/
/*                                                                   */
/*  Name        : Indication::GetWire                                */
/*                                                                   */
/*  Description :                                                    */
/*  Return true if the entry is wired and also copy in the wire      */
/*  details.                                                         */
/*  NOTE: All pointers must be for existing structures               */
/*                                                                   */
/*******************************************************************/
Logical Indication::GetWire(           /* Return true if wired.     */
    SomiError   *error,                /* Error value to return     */
    SomiContext *context,              /* Destination context       */
    SomiValue   *classes,              /* The returned class list   */
    SomiValue   *instances,            /* The returned instances    */
    SomiValue   *action_info );        /* Action info ( crit )      */

/*******************************************************************/
/*                                                                   */
/*  Name        : ProcessActionCnf                                   */
/*                                                                   */
/*  Description :                                                    */
/*  Process the supplied message and if the invoke id is the same    */
/*  as the one in the put away in the FIFO then extract the data     */
/*                                                                   */
/*******************************************************************/
Logical ProcessActionCnf(              /* Return true if valid entry */
    RXMessage   *message );            /* The received message      */

/*******************************************************************/
/*                                                                   */
/*  Name        : Disable                                            */
/*                                                                   */
/*  Description :                                                    */
/*  Process the Logical Disable for this indication. If the entry    */
/*  is matched the indication will be logically disabled and the     */
/*  function will return true.                                       */
/*                                                                   */
/*******************************************************************/
Logical Disable(                       /* Return true if matched    */
    Logical     *clear,                /* True if LD alarm clear.   */
    int         type,                  /* Type of disable.          */
    Logical     disable_ctrl,          /* Disable control           */
    char        *cat_id_mask );        /* Category mask             */

/*******************************************************************/
/*                                                                   */
/*  Name        : Enable                                             */
/*                                                                   */
/*  Description :                                                    */
/*  If there was a prior disable then the disable will be cleared.   */
/*  If the indication was disabled by the configuration then it will*/
/*  cleared and the function will return true.                       */
/*                                                                   */
/*******************************************************************/
Logical Enable(                        /* Return true if matched    */
    Logical     *clear,                /* True if LD alarm clear.   */
    int         type,                  /* Type of disable.          */
    Logical     disable_ctrl,          /* Disable control           */
    char        *cat_id_mask );        /* Category mask             */

/*******************************************************************/
/*                                                                   */
/*  Name        : Default                                            */
/*                                                                   */
/*  Description :                                                    */
/*  If there was a prior disable or enable then the indication will */
/*  be restored to its former self and all associated disable alarms*/
/*  will be cleared                                                  */
/*                                                                   */
/*******************************************************************/
```

```
481  Logical Default(                    /* Return true if matched   */
482      Logical     *clear,             /* True if LD alarm clear.  */
483      int         type,               /* Type of disable,         */
484      Logical     disable_ctrl,       /* Disable control          */
485      char        *cat_id_mask );     /* Category mask            */
486
487  /*****************************************************************/
488  /*                                                               */
489  /*  Name         : SetDisable ( VIRTUAL )                        */
490  /*                                                               */
491  /*  Description :                                                */
492  /*  Instructs LDInd to set Disable flags and generate alarms.    */
493  /*  Simply allows inheriting classes to make use of this function.*/
494  /*                                                               */
495  /*****************************************************************/
496  virtual Logical SetDisable(         /* Return true if matched   */
497      int,                            /* Type of disable function */
498      int )  {                        /* Type of disable.         */
499      return ( BoolFalse );
500  }
501
502  /*****************************************************************/
503  /*                                                               */
504  /*  Name         : ClearDisable ( VIRTUAL )                      */
505  /*                                                               */
506  /*  Description :                                                */
507  /*  Instructs LDInd to clear Disable flags and generate alarms.  */
508  /*  Simply allows inheriting classes to make use of this function.*/
509  /*                                                               */
510  /*****************************************************************/
511  virtual Logical ClearDisable(       /* Return true if matched   */
512      int,                            /* Type of disable function */
513      int )  {                        /* Type of disable.         */
514      return ( BoolFalse );
515  }
516
517  /*****************************************************************/
518  /*                                                               */
519  /*  Name         : FreeAllValues                                 */
520  /*                                                               */
521  /*  Description :                                                */
522  /*  Frees all SomValues supplied.                                */
523  /*                                                               */
524  /*****************************************************************/
525  void FreeAllValues(
526      SomValue   *v1 );               /* SomValue to free         */
527
528  };
529  #endif
```

APPENDIX E: EXAMPLE OF OBJECT-TO-TABLE MAPPING DEFINITION FOR THE DATABASE

Several references have been made in the study to the object-to-table mapping definitions. This was required for the Historian and Configurator subsystems where it was necessary to 'interface' objects to tables of the underlying Relational Database Management System.

The definition file is in the form of an ASCII file - so that the mapping rules are user-definable and not program-coded. Both the Historian and Configurator Object Managers read in the entire definition file into memory on startup - this being done for performance reasons.

The file is divided up into 3 main sections:

(a) The first section is for definition of the class hierarchy (in other words, the Historian or Configurators containment tree).

(b) The second section defines the mapping of object *classes* to RDBMS *tables*.

(c) The third section defines the mapping of object attributes to columns of the relevant RDBMS tables.

Figure E.1 is an example listing of the definition file for the Configurator subsystem.

Figure E.1: Example listing of Object-to-Table Definition File.

```
# This line is for identification by the 'what' command
# @(#)dbdef.conf       3.11 93/02/23 13:28:59 Aslan Av (CONF) (C) BSW-Data (Pty) Ltd.
*************************
* SQL DEFINITION FILE  *
*************************

*****************************************************************************
CLASS HIERARCHY DEFINITION
*****************************************************************************
{
Category        ;
                ;
FlickOption     ;
                ;
GRiceType       ;
                ;
GEquipType      ;
                ;
GEquipType      ,GEquip         ;
*               ,               ;
GEquip          ;
                ;
GEquip          ,GIndication    ;
*               ,               ;
GEquip          ,GIndication    ,Indication;
*               ,*              ,           ;
GEquip          ,Equipment      ;
*               ,               ;
GEquip          ,GIndication    ,GIndRiceOption;
*               ,*              ,               ;
GGraphElmnt     ;
                ;
GOpType         ;
                ;
GSeverity       ;
                ;
GWAL0Type       ;
                ;
GWAL0Type       ,GWAL0WAL1Type  ;
*               ,               ;
GWAL1Type       ;
                ;
GWAL1Type       ,GCPUType       ;
*               ,               ;
GWAL2Type       ;
                ;
GWAL3Type       ;
                ;
GWAL3Type       ,GWAL3IndType   ;
*               ,               ;
GWAL            ;
                ;
GWAL            ,GWALFault      ;
*               ,               ;
Responsibility;
                ;
Responsibility,LegSet           ;
*               ,               ;
Location        ;
                ;
```

```
LegSet      ;
            ;
LegSet      ,Leg          ;
*           ,             ;
Leg         ;
            ;
Leg         ,WAL0         ;
*           ,             ;
Leg         ,LocSequence  ;
*           ,             ;
Leg         ,Equipment    ;
*           ,             ;
Leg         ,Affects      ;
*           ,             ;
Leg         ,AffectedBy   ;
*           ,             ;
Leg         ,Equipment    ,Indication ;
*           ,*            ,           ;
Leg         ,Equipment    ,Indication ,UpLink      ;
*           ,*            ,*          ,            ;
Leg         ,Equipment    ,Indication ,DownLink    ;
*           ,*            ,*          ,            ;
Leg         ,Equipment    ,Indication ,WireL       ;
*           ,*            ,*          ,            ;
Leg         ,Equipment    ,Indication ,IndRiceOption;
*           ,*            ,*          ,            ;
WAL0        ;
            ;
WAL0        ,WAL1         ;
*           ,             ;
WAL0        ,WAL1         ,WAL2         ;
*           ,*            ,             ;
WAL0        ,WAL1         ,WAL2         , WAL3      ;
*           ,*            ,*            ,           ;
WAL0        ,WAL1         ,WAL2         , WAL3      ,WAL4      ;
*           ,*            ,*            ,*          ,          ;
WAL0        ,WAL1         ,WAL2         , WAL3      ,WAL4      ,WAL5      ;
*           ,*            ,*            ,*          ,*         ,          ;
WAL0        ,WAL1         ,WAL2         , WAL3      ,WAL4      ,WAL5      ,WireP ;
*           ,*            ,*            ,*          ,*         ,*         ,      ;
Top         ;
            ;
Top         ,Category     ;
            ,             ;
Top         ,FlickOption  ;
            ,             ;
Top         ,GRiceType    ;
            ,             ;
Top         ,GEquipType   ;
            ,             ;
Top         ,GOpType      ;
            ,             ;
Top         ,GEquip       ,GIndication   ;
            ,*            ,              ;
Top         ,GEquip       ,GIndication   ,GIndRiceOption;
            ,*            ,*             ,              ;
Top         ,GWAL0Type    ;
            ,             ;
Top         ,GWAL0Type    ,GWAL0WAL1Type ;
            ,*            ,              ;
```

```
Top             ,GWAL1Type      ;
                ,               ;
Top             ,GWAL2Type      ;
                ,               ;
Top             ,GWAL3Type      ;
                ,               ;
Top             ,GWAL           ;
                ,               ;
Top             ,Responsibility;
                ,               ;
Top             ,Location       ;
                ,               ;
Top             ,LegSet         ;
                ,               ;
Top             ,LegSet         ,Leg            ;
                ,*              ,               ;
Top             ,WALO           ;
                ,               ;
Top             ,WALO           ,WAL1       ;
                ,*              ,           ;
Top             ,WALO           ,WAL1       ,WAL2           ;
                ,*              ,*          ,               ;
Top             ,WALO           ,WAL1       ,WAL2           , WAL3      ;
                ,*              ,*          ,*              ,           ;
}
*****************************************************************************
CLASS/ PRIMARY TABLE DEFINITION
*****************************************************************************
CLASS           TABLE           CLASSID
{
Top             ,                       ,top;
Category        ,CATEGORY               ,cat_id;
Equipment       ,EQUIPMENT              ,eqt_id ;
FlickOption     ,FLICK_OPTION           ,fl_option;
GRiceType       ,G_RICE_TYPE            ,rice_type;
GEquip          ,G_EQUIP                ,code;
GEquipType      ,G_EQUIP_TYPE           ,type;
GGraphElmnt     ,G_GRAPHICS_ELEMENT,graph_elmnt;
GOpType         ,G_OP_TYPE              ,op_type;
GSeverity       ,G_SEVERITY             ,severity;
GWALOType       ,G_WALO_TYPE            ,walO_type;
GWAL1Type       ,G_WAL1_TYPE            ,wal1_type;
GWALOWAL1Type ,G_WALO_WAL1_TYPE   ,wal1_type;
GCPUType        ,G_CPU_TYPE             ,cpu_type;
GWAL2Type       ,G_WAL2_TYPE            ,wal2_type;
GWAL3Type       ,G_WAL3_TYPE            ,wal3_type;
GWAL3IndType    ,G_WAL3_IND_TYPE        ,ind_type;
GIndication     ,G_INDICATION           ,ind_id;
GIndRiceOption,G_IND_RICE_OPTION  ,rice_type;
GWAL            ,G_WAL                  ,wal_level;
GWALFault       ,G_WAL_FAULT            ,fault_id;
Indication      ,INDICATION             ,ind_id ;
IndRiceOption ,IND_RICE_OPTION    ,rice_type;
UpLink          ,IND_LINK               ,out_attr;
DownLink        ,IND_LINK               ,in_attr;
Leg             ,LEG                    ,leg_id;
Affects         ,LEG_LINK               ,affects_leg_id;
AffectedBy      ,LEG_LINK               ,affected_by_leg_id;
LegSet          ,LEGSET                 ,l_s_id;
Location        ,LOCATION               ,loc_id;
```

```
LocSequence     ,LOC_SEQUENCE       ,seq_no;
Responsibility,RESPONSIBILITY       ,resp_code;
WAL0            ,WAL0               ,wal0_num;
WAL1            ,WAL1               ,wal1_num;
WAL2            ,WAL2               ,wal2_num;
WAL3            ,WAL3               ,wal3_num;
WAL4            ,WAL4               ,wal4_num;
WAL5            ,WAL5               ,wal5_num;
WireL           ,WIRE               ,crit_id;
WireP           ,WIRE               ,crit_id;
)
*************************************************************************
ATTRIBUTE/COLUMN DEFINITION
*************************************************************************
ACTIONCLASS     BASECLASS       ATTRIBUTE       TYPE            TABLE           COLUMN          TYPE
(
Category,       Category,       cat_id,         OBJ_STRING,     CATEGORY,       CAT_ID,         CHAR;
Category,       Category,       cat_type,       OBJ_STRING,     CATEGORY,       CAT_TYPE,       CHAR;
Category,       Category,       cat_desc,       OBJ_STRING,     CATEGORY,       CAT_DESC,       CHAR;
Category,       Category,       alarm_label,    OBJ_STRING,     CATEGORY,       ALARM_LABEL,    CHAR;
Category,       Category,       urgency,        OBJ_SHORT,      CATEGORY,       URGENCY,        NUMBER;
Category,       Category,       severity,       OBJ_STRING,     CATEGORY,       SEVERITY,       CHAR;
Category,       Category,       graph_elmnt,    OBJ_STRING,     CATEGORY,       GRAPH_ELMNT,    CHAR;
Equipment,      Equipment,      eqt_id,         OBJ_STRING,     EQUIPMENT,      EQT_ID,         CHAR;
Indication,     Equipment,      eqt_id,         OBJ_STRING,     INDICATION,     EQT_ID,         CHAR;
IndRiceOption,  Equipment,      eqt_id,         OBJ_STRING,     IND_RICE_OPTION,EQT_ID,         CHAR;
UpLink,         Equipment,      eqt_id,         OBJ_STRING,     IND_LINK,       EQT_ID,         CHAR;
DownLink,       Equipment,      eqt_id,         OBJ_STRING,     IND_LINK,       UP_EQT_ID,      CHAR;
WireL,          Equipment,      eqt_id,         OBJ_STRING,     WIRE,           EQT_ID,         CHAR;
Equipment,      Equipment,      description,    OBJ_STRING,     EQUIPMENT,      DESCRIPTION,    CHAR;
Equipment,      Equipment,      frequency,      OBJ_STRING,     EQUIPMENT,      FREQUENCY,      CHAR;
Equipment,      Equipment,      sequence_no,    OBJ_SHORT,      EQUIPMENT,      SEQUENCE_NO,    NUMBER;
Equipment,      Equipment,      code,           OBJ_STRING,     EQUIPMENT,      CODE,           CHAR;
Equipment,      Equipment,      loc_id,         OBJ_STRING,     EQUIPMENT,      LOC_ID,         CHAR;
Equipment,      Equipment,      graph_elmnt,    OBJ_STRING,     EQUIPMENT,      GRAPH_ELMNT,    CHAR;
FlickOption,    FlickOption,    fl_option,      OBJ_LONG ,      FLICK_OPTION,   FL_OPTION,      NUMBER;
FlickOption,    FlickOption,    set_fl_t_o,     OBJ_SHORT,      FLICK_OPTION,   SET_FL_T_O,     NUMBER;
FlickOption,    FlickOption,    reset_fl_t_o,   OBJ_SHORT,      FLICK_OPTION,   RESET_FL_T_O,   NUMBER;
FlickOption,    FlickOption,    fl_thresh,      OBJ_FLOAT,      FLICK_OPTION,   FL_THRESH,      FLOAT;
FlickOption,    FlickOption,    fl_decay,       OBJ_SHORT,      FLICK_OPTION,   FL_DECAY,       NUMBER;
FlickOption,    FlickOption,    fl_desc,        OBJ_STRING,     FLICK_OPTION,   FL_DESC,        CHAR;
GRiceType,      GRiceType,      rice_type,      OBJ_STRING,     G_RICE_TYPE,    RICE_TYPE,      CHAR;
GRiceType,      GRiceType,      description,    OBJ_STRING,     G_RICE_TYPE,    DESCRIPTION,    CHAR;
GEquip,         GEquip,         code,           OBJ_STRING,     G_EQUIP,        CODE,           CHAR;
Equipment,      GEquip,         code,           OBJ_STRING,     EQUIPMENT,      CODE,           CHAR;
GIndication,    GEquip,         code,           OBJ_STRING,     G_INDICATION,   CODE,           CHAR;
GIndRiceOption,GEquip,          code,           OBJ_STRING,     G_IND_RICE_OPTION,CODE,         CHAR;
GEquip,         GEquip,         type,           OBJ_STRING,     G_EQUIP,        TYPE,           CHAR;
GEquip,         GEquip,         description,    OBJ_STRING,     G_EQUIP,        DESCRIPTION,    CHAR;
GEquip,         GEquip,         manufacturer,   OBJ_STRING,     G_EQUIP,        MANUFACTURER,   CHAR;
GEquip,         GEquip,         power,          OBJ_STRING,     G_EQUIP,        POWER,          CHAR;
GEquip,         GEquip,         model,          OBJ_STRING,     G_EQUIP,        MODEL,          CHAR;
GEquip,         GEquip,         graph_elmnt,    OBJ_STRING,     G_EQUIP,        GRAPH_ELMNT,    CHAR;
GEquipType,     GEquipType,     type,           OBJ_STRING,     G_EQUIP_TYPE,   TYPE,           CHAR;
GEquip,         GEquipType,     type,           OBJ_STRING,     G_EQUIP,        TYPE,           CHAR;
GEquipType,     GEquipType,     description,    OBJ_STRING,     G_EQUIP_TYPE,   DESCRIPTION,    CHAR;
GGraphElmnt,    GGraphElmnt,    graph_elmnt,    OBJ_STRING,     G_GRAPHICS_ELEMENT,GRAPH_ELMNT, CHAR;
GGraphElmnt,    GGraphElmnt,    elmnt_val,      OBJ_LONG,       G_GRAPHICS_ELEMENT,ELMNT_VAL,   NUMBER;
GGraphElmnt,    GGraphElmnt,    description,    OBJ_STRING,     G_GRAPHICS_ELEMENT,DESCRIPTION, CHAR;
GOpType,        GOpType,        op_type,        OBJ_STRING,     G_OP_TYPE,      OP_TYPE,        CHAR;
```

APPENDIX F: CLASS EXAMPLES

Appendix F contains examples of class definitions as referred to in this report, in the form of abbreviated listings of the C++ include files. The filenames are: *DktSummary.h*, *MimDisplay.h*, *LinePerfDisp.h*, *WAL.h* and *Docket.h*.

Printed by revision 3.4 of lprt.e
File Name : DktSummary.h Date : Wed Jan 27 11:45:28 1993 Page : 1

```
 1  /* This line is for identification by the 'what' command! */
 2  /* @(#)DktSummary.h 3.1 93/01/19 05:36:57 Aslan AV DMI 1 (C) BSW-Data (Pty) Ltd.
 3  */
 4  /*****************************************************************/
 5  /*                                                               */
 6  /*  TITLE BLOCK:                                                 */
 7  /*  ============                                                 */
 8  /*                                                               */
 9  /*  Name        : DktSummary.h                                   */
10  /*                                                               */
11  /*  I.D. Number : ASLAN      Revr 0.00     Rel: 0.00             */
12  /*                                                               */
13  /*    Software Developed By : BSW-DATA                           */
14  /*                            Private Bag X35                    */
15  /*                            Halfway House                      */
16  /*                            1685                               */
17  /*  Function    : This is the interface for the DocketSummary class.*/
18  /*                                                               */
19  /*  History     :                                                */
20  /*                                                               */
21  /*          Event              Date       Author      Approved   */
22  /*                                                               */
23  /*    Specification          92/03/19      PGF          XXX      */
24  /*    Design                 92/03/19      PGF          XXX      */
25  /*    Implementation         92/03/19      PGF          XXX      */
26  /*    Audit                  XX/XX/XX                   XXX      */
27  /*                                                               */
28  /*                                                               */
29  /*  Revisions:  Revision     Date       Author      Approved     */
30  /*                0.00       XX/XX/XX     XXX         XXX      */
31  /*                                                               */
32  /*  Revision Description:                                        */
33  /*  =====================                                        */
34  /*    0.00 - None                                                */
35  /*                                                               */
36  /*                                                               */
37  /*  Functional Specification:                                    */
38  /*  =========================                                    */
39  /*                                                               */
40  /*  Provides support for docket summaries in either the list of  */
41  /*  dockets controlled by the operator position or dockets in    */
42  /*  any general display list.                                    */
43  /*                                                               */
44  /*****************************************************************/
45
46  #ifndef DOCKETSUMMARY_INCLUDED
47  #define DOCKETSUMMARY_INCLUDED
48
49  #include    "ON.h"
50  #include    "RtomTypes.h"
51
52  /*****************************************************************/
53  /*  Class name: DocketSummary                                    */
54  /*  Description: This class contains the summary information for */
55  /*      docket. It also provides methods for the manipulation of */
56  /*      dockets. Any                                             */
57  /*      updates to the docket summary information are done by RTOM, */
58  /*      with the exception of changes to the docket's category   */
59  /*      filter or delay, which are controlled internally in the RMU.*/
60  /*                                                               */
61  /*****************************************************************/
62  class   DocketList;
63  class   DocketSummary
64      : public LinkList
65  {
66  public:
67  /*****************************************************************/
68  /*  Function name:  DocketSummary (constructor)                  */
69  /*  Description:    Creates an instance from the given list of   */
70  /*      attributes and values. A matched pair of records gives the */
71  /*      identifiers and values.                                  */
72  /*                                                               */
73  /*****************************************************************/
74      DocketSummary   (SomiValue    *rec   /*  attribute records    */
75                      ,SomiContext *rtom  /*  owning RTOM          */
76                      ,SomiError   *err); /*  error return         */
77
78  /*****************************************************************/
79  /*  Function name : ~DocketSummary                               */
80  /*  Description :   The destructor is invoked as a result of an  */
81  /*      M_SET (REMOVE_VAL) to the parent DocketList.             */
82  /*                                                               */
83  /*****************************************************************/
84      ~DocketSummary ();
85
86  /*****************************************************************/
87  /*  Function name : SetValues                                    */
88  /*  Description :   This method handles SET_IND (INDVAL) to alter */
89  /*      the values of attributes.                                */
90  /*                                                               */
91  /*****************************************************************/
92      SomiError   SetValues
93          (SomiValue  *attributes);       /*  attribute records    */
94
95      /*  Attributes  */
96      SomiError DoSet
97          (SomiValue  *attributes);       /*  attribute records    */
98
99      /* Attribute values */
100     char    dref [DREF_SIZE + 1];
101     long    docket_line;
102     char    leg_id [LEG_ID_SIZE + 1];
103     char    cat_list [MAX_CATEGORIES + 1];
104     short   urgency;
105     char    opid [OPID_SIZE + 1];
106     char    f_loc [F_LOC_SIZE + 1];
107     long    d_flags;
108     char    progress_code [PCODE_SIZE + 1];
109     short   l_s_id;
110
111     /* sublist identifier (A,B,F,P for Nimrac) */
112     char    sublist;
113
114     /* category filter  */
115     char        cat_filter [MAX_CATEGORIES + 1];
116     static char def_cat_filter [MAX_CATEGORIES + 1];
117     char        prev_cat [MAX_CATEGORIES + 1];
118
119
```

```
120       /*  docket timer    */
121       void    *timer;     /* active timer                          */
122       int     delaySet;   /* indicates whether delay was set */
123       int     delay;      /* timeout delay for docket              */
124       time_t  delayExpiry; /* end of timeout period                */
125
126       /*  owner node  */
127       SoafContext rtos;
128
129   private:
130       DocketSummary   ();         /*  hide the default constructor    */
131   };
132
133
134   #endif  /* DOCKETSUMMARY_INCLUDED   */
```

Printed by revision 3.4 of lprt.c
File Name : MimDisp.h Date : Wed Jan 27 11:43:45 1993 Page : 1

```
/* This line is for identification by the 'what' command! */
/* C(#)MimDisp.h    3.2 93/01/25 12:33:44 Aslan Av [MMI] (C) BSW-Data (Pty) Ltd. */
/
/*****************************************************************************/
/*                                                                           */
/*  TITLE BLOCK:                                                             */
/*  ============                                                             */
/*                                                                           */
/*  Name         : MimDisplay.h                                              */
/*                                                                           */
/*  I.D. Number : ASLAV  Rev: 0.00  Rel: 0.00                                */
/*                                                                           */
/*  Software Developed By : BSW-DATA                                         */
/*                          Private Bag X35                                  */
/*                          Halfway House                                    */
/*                          1685                                             */
/*                                                                           */
/*  Function     : This is the interface for the                             */
/*                 MimacDisplay class.                                       */
/*                                                                           */
/*                                                                           */
/*  History      :                                                           */
/*                                                                           */
/*  Event                Date        Author      Approved                    */
/*                                                                           */
/*  Specification        92/03/23    PGF         XXX                         */
/*  Design               92/03/23    PGF         XXX                         */
/*  Implementation       92/03/23    PGF         XXX                         */
/*  Audit                                                                    */
/*                                                                           */
/*                                                                           */
/*  Revisions:  Revision     Date     Author      Approved                    */
/*              0.00         XX/XX/XX   XXX       XXX                         */
/*                                                                           */
/*  Revision Description:                                                    */
/*  =====================                                                    */
/*  0.00 - None                                                              */
/*                                                                           */
/*                                                                           */
/*  Functional Specification:                                                */
/*  =========================                                                */
/*                                                                           */
/*  Implements the general form of all Mimac displays.                       */
/*                                                                           */
/*****************************************************************************/

#ifndef MIMMACDISPLAY_INCLUDED
#define MIMMACDISPLAY_INCLUDED

#ifdef CUI
#   include "CUIDspxxG.h"
#else
#   include "DspxxG.h"
#endif

/*****************************************************************************/
/*  Class name: MimacDisplay                                                 */
/*  Description: This class implements the basic Mimac text display.        */
/*       All Mimac displays have two header lines, the first containing      */
/*       the display title, the operator position name, the node number      */
/*       on which the MMI exists, the user name, and the date and time.      */
/*       The second header line contains information pertinent to the        */
/*       display. There is also a top-of-display marker line and a           */
/*       bottom-of-display marker line. The sort keys of the top and         */
/*       bottom markers are set to 'A' and 'Z' respectively.                 */
/*       Displays derived from this class can set up the sort keys for       */
/*       lines in the display according to their needs.                      */
/*                                                                           */
/*****************************************************************************/
class  MMITop;
class  MimacDisplay:
    public  Gt_MappedWindowG
{
public:
/*****************************************************************************/
/*  Function name : MimacDisplay                                             */
/*  Description :   The constructor for this class will invoke the           */
/*       base class constructor to create the mapped window. It will         */
/*       then set up the first header line and the top and bottom            */
/*       marker lines.                                                       */
/*                                                                           */
/*****************************************************************************/
    MimacDisplay   ( Gt_Display       &display
                   , char             *windowTitle
                   , char             *displayTitle
                   , int              headerFlds
                   , int              lineFields
                   , MMITop           *top );

/*****************************************************************************/
/*  Function name : ~MimacDocketDisplay                                      */
/*  Description :   The destructor for this class will shut down             */
/*       the display.                                                        */
/*                                                                           */
/*****************************************************************************/
    ~MimacDisplay  ();

/*****************************************************************************/
/*  Function name: CloseDisplay                                              */
/*  Description: This is a virtual function which is activated when          */
/*       the Gt_MappedWindowG's close button is clicked. The base            */
/*       class function does nothing - displays derived from this            */
/*       class can specify other functionality.                              */
/*                                                                           */
/*****************************************************************************/
    virtual void CloseDisplay () {;}

/*****************************************************************************/
/*  Function name: ReportError                                               */
/*  Description: Raises an error dialog to inform the user.                  */
/*       ('printf'-style argument list)                                      */
/*                                                                           */
/*****************************************************************************/
    void ReportError (char *format, ...);

private:
    Gt_MappedLineG      topMark;        /*  top of display marker      */
    Gt_MappedLineG      botMark;        /*  bottom of display marker   */
    Gt_MappedFieldG     topField;       /*  top line single field      */
    Gt_MappedFieldG     botField;       /*  bot line single field      */
```

```
120
121     /* Reacts to Window close */
122     static void CloseCallback (void *ud, void *cd);
123
124     /* Reacts to error dialog close */
125     static void ErrorCloseCB   (void *ud, void *cd);
126 };
127
128 #endif /* MINIACDISPLAY_INCLUDED */
```

Printed by revision 3.4 of lprt.c
File Name : linePerfDisp.h Date : Wed Jan 27 11:51:27 1993 Page : 1

```
/* This line is for identification by the 'what' command! */
/* @(#)LinePerfDisp.h   3.2 93/01/25 12:34:08 Aslan Gv DMI 1 (C) BSV-Data (Pty) L
td. */

/*****************************************************************/
/*                                                               */
/*  TITLE BLOCK:                                                 */
/*  ============                                                 */
/*                                                               */
/*  Name        : LinePerfDisp.h                                 */
/*                                                               */
/*  I.D. Number : ASLAN  Rev: 0.00  Rel: 0.00                    */
/*                                                               */
/*  Software Developed By : BSV-DATA                             */
/*                          Private Bag X35                      */
/*                          Halfway House                        */
/*                          1685                                 */
/*                                                               */
/*  Function    :                                                */
/*                                                               */
/*                                                               */
/*  History     :                                                */
/*                                                               */
/*  Event               Date        Author      Approved         */
/*                                                               */
/*  Specification       92/12/04    CK          XXX              */
/*  Design              92/12/04    CK          XXX              */
/*  Implementation      92/12/04    CK          XXX              */
/*  Audit                                                        */
/*                                                               */
/*                                                               */
/*  Revisions:  Revision     Date     Author      Approved       */
/*              0.00         XX/XX/XX    XXX      XXX            */
/*                                                               */
/*  Revision Description:                                        */
/*  =====================                                        */
/*  0.00 - None                                                  */
/*                                                               */
/*                                                               */
/*  Functional Specifications:                                   */
/*  ==========================                                   */
/*                                                               */
/*                                                               */
/*****************************************************************/

#ifndef LINEPERFDISP_INCLUDED
#define LINEPERFDISP_INCLUDED

/* Include files */
#include "DM.h"
#include "MMITop.h"
#include "MimDisp.h"

#ifdef __cl
#   include "CliDspxG.h"
#else
#   include "DspxG.h"
#endif


/* Define the maximum number of outstations */
#define MAX_OUTSTATIONS   20

/* Define length for the display identifier */
#define ID_LENGTH         45

/* Determine number of profile data */
#define STLP_PROFILE_LINE_NUM      6

/* Define sort keys */
#define STLP_RICE_INFO_SORT        'G'
#define STLP_MINUTE_HEADING_SORT   'H'
#define STLP_HOURLY_HEADING_SORT   'H'

/* Determine the colum positions */
#define STLP_OUTSTATION_POS        0
#define STLP_LOCATION_POS          3
#define STLP_START_DATA_POS        8
#define STLP_DATA_DISTANCE         5

/* The maximum number of lines */
#define STLP_MAX_LINES             50

#define LOCATION_SIZE              4
#define MAX_DATA                   24

typedef struct {
   short      outstation_num;
   char       location[LOCATION_SIZE];
   short      profile_data[MAX_DATA];
   int        elements;

} LinePerformance;

/* Line performance data type */
typedef enum {
   HOURS = 0,
   MINUTES
} STLPDataType;

/*****************************************************************/
/*  Class  : linePerfDisplay                                     */
/*  Description : This class is a container for the detail lines of the   */
/*       Line Performance Display.  When the Line Performance Display is  */
/*       selected by the operator, an instance of this class is created in */
/*       the MMI's object tree, and an M_ACTION request is sent to Rtom to */
/*       notify it the details are required at the MMI.          */
/*       The Rtom will then generate an M_SET (replace) directed at this   */
/*       instance for all outstation line performance information.       */
/*       This display is not active.                             */
/*****************************************************************/
class MMITop;
class LinePerfDisplay : public CMObject,
                        public MimicDisplay
{
private :
   static Logical          firstInstance;
   static ClassBluePrint   *bluePrint;

public :
```

```
120  /*****************************************************************/
121  /*   Function name :  LinePerfDisplay (constructor)              */
122  /*   Description   : This constructs the Line Performance Display.  It   */
123  /*      notifies Rtcm of existence via an M_ACTION request to the VALD   */
124  /*      in the Rtcm.  This instance is created as a child of the MMI root */
125  /*      object.                                                  */
126  /*****************************************************************/
127     LinePerfDisplay(SomiError      *error,
128                     MMITop         *father,
129                     ClassEntry     *cl_entry,
130                     SomiValue      *id,
131                     SomiValue      *value,
132                     OMObject       *refObj,
133                     RXMessage      *rx_msg,
134                     RspBlock       *resp);
135
136  /*****************************************************************/
137  /*   Function name :  ~LinePerfDisplay (destructor)              */
138  /*   Description   : The destructor will be invoked when the Line        */
139  /*      Performance Display is closed.                           */
140  /*****************************************************************/
141     ~LinePerfDisplay()  {;}
142
143  /*****************************************************************/
144  /*   Function name :  SetInd                                     */
145  /*   Description   : This member is invoked when a SET_IND is received for*/
146  /*      the display.                                             */
147  /*****************************************************************/
148     void SetInd(RXMessage      *rx_msg,
149                 RspBlock       *resp,
150                 SomiValue      *attrMod_lst);
151
152  /*****************************************************************/
153  /*   Function name :  ActionCnf                                  */
154  /*   Description   : handles the response to the ActionReq issued by the */
155  /*      constructor.                                             */
156  /*****************************************************************/
157     void ActionCnf(RXMessage      *rx_msg);
158
159  /*****************************************************************/
160  /*   Function name :  FormatAndSendActionReq                     */
161  /*   Description   : Puts together the action request and issues it.     */
162  /*****************************************************************/
163     void FormatAndSendActionReq(SomiContext   *dest,
164                                 InvokeID      *invokeId);
165
166  /*****************************************************************/
167  /*   Function name :  SetupActionInfo                            */
168  /*   Description   : Assembles the action information SomiValue          */
169  /*****************************************************************/
170     void SetupActionInfo(SomiValue      *AI);
171
172  /*****************************************************************/
173  /*   Function name :  CloseDisplay                               */
174  /*   Description   : This invokes the constructor,               */
175  /*****************************************************************/
176     virtual void CloseDisplay();
177
178  /*****************************************************************/
179  /*   Function name :  ExtractPerformance                         */
180  /*   Description   : This extracts the performance values from the        */
181  /*      record and stores them in the member structure.           */
182  /*****************************************************************/
183     SomiError ExtractPerformance(SomiValue         *somiVal,
184                                  LinePerformance   *perf_ptr);
185
186  /*****************************************************************/
187  /*   Function name :  FormatDisplayLine                          */
188  /*   Description   : Formats the data in the required format for display  */
189  /*      purposes.                                                */
190  /*****************************************************************/
191     void FormatDisplayLine(LinePerformance      *linePerf_ptr,
192                            Gt_MappedLineG       *line_ptr,
193                            Gt_MappedFieldG      *field_ptr);
194
195  /*****************************************************************/
196  /*   Function name :  AddFieldToLine                             */
197  /*   Description   : Adds a field to the line.                   */
198  /*****************************************************************/
199     void AddFieldToLine(Gt_MappedLineG      *line_ptr,
200                         Gt_MappedFieldG     *field_ptr,
201                         char                *fieldText);
202
203     /* Tube and sequence numbers */
204     static int      nextTube;       /* Increments to give new tube */
205     int             tube;           /* Tube for this display */
206     int             seq;            /* Next sequence expected */
207
208     /* Creation status */
209     int             created;        /* 1 = successfully created */
210
211     /* Line performance structure */
212     LinePerformance     minute_profile[MAX_OUTSTATIONS * 2];
213     int                 minute_profile_count;
214     LinePerformance     hourly_profile[MAX_OUTSTATIONS * 2];
215     int                 hourly_profile_count;
216     short               route_num;
217     short               node_num;
218     long                release_num;
219     long                release_date;
220     long                startup_date;
221     short               startup_node_num;
222
223     /* Display information */
224     MMITop                  *myTop;
225     char                    display_name[DISPLAY_ID_LENGTH];
226     STLPDataType            type;
227     Gt_MappedWindowG        window;
228     Gt_MappedLineG          line[MAX_OUTSTATIONS * 2];
229     Gt_MappedFieldG         field[MAX_OUTSTATIONS * 2];
230
231  protected :
232  /*****************************************************************/
233  /*   Function name :  DisplayRICEInfo                            */
234  /*   Description   : Displays the RICE information.              */
235  /*****************************************************************/
236     void DisplayRICEInfo();
237
238  /*****************************************************************/
239  /*   Function name :  DisplayHourlyProfile                       */
```

```
240   /*   Description   : Displays a line of hourly profile data.          */
241   /*************************************** ***************************************/
242     void DisplayHourlyProfile(LinePerformance   *performance_ptr);
243
244   /*************************************************************************/
245   /*   Function name :  DisplayHourlyProfileHeading                       */
246   /*   Description   : Displays the headings for the hourly profile data.  */
247   /*************************************************************************/
248     void DisplayHourlyProfileHeading();
249
250   /*************************************************************************/
251   /*   Function name :  DisplayMinuteProfile                              */
252   /*   Description   : Displays a line of minute profile data.            */
253   /*************************************************************************/
254     void DisplayMinuteProfile(LinePerformance   *performance_ptr);
255
256   /*************************************************************************/
257   /*   Function name :  DisplayMinuteProfileHeading                       */
258   /*   Description   : Displays the headings for the minute profile data.  */
259   /*************************************************************************/
260     void DisplayMinuteProfileHeading();
261
262   } ; /* class LinePerfDisplay */
263
264   #endif /* LINEPERFDISP_INCLUDED */
```

Printed by revision 3.4 of lprt.c
File Name : WAL.h Date : Wed Jan 27 08:10:25 1993 Page 1

```
/* This line is for identification by the 'what' command! */
/* %W% %E% %U% Asian Av (RIOH) (C) BSW-Data (Pty) Ltd. */
/*****************************************************************/
/*                                                               */
/*  TITLE BLOCK:                                                 */
/*  ============                                                 */
/*                                                               */
/*  Name        : WAL.h                                          */
/*                                                               */
/*  I.D. Number : ASLAN     Rev: 0.00     Rel: 0.00              */
/*                                                               */
/*    Software Developed By : BSW-DATA                           */
/*                            Private Bag X35                    */
/*                            Halfway House                      */
/*                            1685                               */
/*                                                               */
/*  Function    : Class definition for WAL.                      */
/*                                                               */
/*  History     :                                                */
/*                                                               */
/*          Event              Date      Author      Approved    */
/*                                                               */
/*    Specification          92/09/01      PBB          AB       */
/*    Design                 92/09/01      PBB          AB       */
/*    Implementation         92/09/01      PBB          AB       */
/*    Audit                  XX/XX/XX                   XXX      */
/*                                                               */
/*  Revisions:  Revision     Date      Author      Approved      */
/*                0.00      XX/XX/XX     XXX          XXX       */
/*                                                               */
/*  Revision Description:                                        */
/*  =====================                                        */
/*    0.00 - None                                                */
/*                                                               */
/*                                                               */
/*  Functional Specification:                                    */
/*  =========================                                    */
/*                                                               */
/*  Definition for the WAL class                                 */
/*                                                               */
/*****************************************************************/
#ifndef WAL_INCLUDED
#define WAL_INCLUDED
#include "OH.h"

/*****************************************************************/
/*                                                               */
/*  Class Name  : WAL                                            */
/*                                                               */
/*  Description :                                                */
/*  This class forms the base class of the World Access Level    */
/*  hierarchy. It allows the configuration process to do its thing */
/*  without having to know what level or class it is dealing with. */
/*                                                               */
/*****************************************************************/
class WAL : public CHObject {

    private:

    public:

/*****************************************************************/
/*                                                               */
/*  Name        : WAL::WAL                                       */
/*                                                               */
/*  Description :                                                */
/*  The constructor for this CHObject                            */
/*                                                               */
/*****************************************************************/
    WAL ( );

/*****************************************************************/
/*                                                               */
/*  Name        : WAL::~WAL                                      */
/*                                                               */
/*  Description :                                                */
/*  The destructor for this CHObject                             */
/*                                                               */
/*****************************************************************/
    ~WAL();

/*****************************************************************/
/*                                                               */
/*  Name        : DoUpdate ( virtual )                           */
/*                                                               */
/*  Description :                                                */
/*  This virtual function allows DoUpdates to be treated generically*/
/*                                                               */
/*****************************************************************/
virtual void DoUpdate(
    short       id,                     /* The wal id number     */
    Logical     flag,                   /* Forced update flag    */
    long        release_num );          /* The latest release #  */

/*****************************************************************/
/*                                                               */
/*  Name        : ConfigStatus ( virtual )                       */
/*                                                               */
/*  Description :                                                */
/*  This virtual function allows the config status to be established*/
/*                                                               */
/*****************************************************************/
virtual Logical ConfigStatus            /* Return true if busy   */
    void );

/*****************************************************************/
/*                                                               */
/*  Name        : StopConfig   ( virtual )                       */
/*                                                               */
/*  Description :                                                */
/*  This virtual function allows the configuration to be stopped */
/*                                                               */
/*****************************************************************/
virtual void StopConfig( void );

/*****************************************************************/
/*                                                               */
/*  Name        : SendConfig                                     */
```

```
121  /*                                                                  */
122  /*  Description :                                                   */
123  /*  Send the current configuration to the specified dest context.   */
124  /*                                                                  */
125  /*******************************************************************/
126  SomIError SendConfig(                  /* Return error if failed.  */
127      Logical      *wal5_valid,          /* Return true if pt wired  */
128      short        wal1_Id,              /* WAL1 id value            */
129      SomIContext *context,              /* The destination context  */
130      Logical      *last_config,         /* Last configured level    */
131      int          level,                /* Current level            */
132      Logical      incremental );        /* Incremental update       */
133
134  /*******************************************************************/
135  /*                                                                  */
136  /*  Name        : SendCreateAction ( virtual )                      */
137  /*                                                                  */
138  /*  Description :                                                   */
139  /*  Tell derived classes to send the configuration data.            */
140  /*                                                                  */
141  /*******************************************************************/
142  virtual SomIError SendCreateAction(    /* Return error if failed   */
143      Logical *wal5_valid,               /* WAL5 entry valid flag    */
144      short    wal1_Id,                  /* WAL1 id value            */
145      SomIContext *context );            /* Destination context      */
146
147  };
148  #endif
```

Printed by revision 3.4 of lprt.c
File Name : Docket.h Date : Wed Jan 27 08:25:50 1993 Page : 1

```
/* This line is for identification by the 'what' command( */
/* %W% %E% %U% Astan Av BETOKI (C) BSW-Data (Pty) Ltd. */
/*****************************************************************/
/*                                                               */
/*  TITLE BLOCK:                                                 */
/*  ============                                                 */
/*                                                               */
/*  Name        : Docket.h                                       */
/*                                                               */
/*  I.D. Number : ASLAW     Rev: 0.00     Rel: 0.00              */
/*                                                               */
/*    Software Developed By : BSW-DATA                           */
/*                            Private Bag X35                    */
/*                            Halfway House                      */
/*                            1685                               */
/*                                                               */
/*  Function    : The Docket handling Object.                    */
/*                                                               */
/*  History     :                                                */
/*                                                               */
/*          Event              Date      Author      Approved    */
/*                                                               */
/*    Specification         92/03/30      PBB          XXX       */
/*    Design                XX/XX/XX      XXX          XXX       */
/*    Implementation        XX/XX/XX      XXX          XXX       */
/*    Audit                 XX/XX/XX                   XXX       */
/*                                                               */
/*                                                               */
/*  Revisions:  Revision     Date       Author      Approved     */
/*                0.00       XX/XX/XX   XXX           XXX       */
/*                                                               */
/*  Revision Description:                                        */
/*  =====================                                        */
/*    0.00 - None                                                */
/*                                                               */
/*                                                               */
/*  Functional Specification:                                    */
/*  =========================                                    */
/*                                                               */
/*  This class provides for the creation/manipulation of fault logs */
/*  ( called dockets ). The actual creation and manipulation of  */
/*  these dockets is performed by the LogicalEquipmentGroup Object */
/*  in RTOK.                                                     */
/*                                                               */
/*****************************************************************/

#ifndef DOCKET_INCLUDED
#define DOCKET_INCLUDED

#include "DocketLink.h"     /* Object manager goodies.           */
#include "RtomTypes.h"      /* Typedefs and other stories        */
#include "Alarm.h"          /* Alarm class                       */
#include "Note.h"           /* Note class                        */
#include "ClearCode.h"      /* ClearCode class                   */
#include "Detail.h"         /* Detail class                      */
#include "Leg.h"            /* Leg class                         */
#include "LegSet.h"         /* The LegSet class.                 */
#include "ScEntry.h"        /* The Scenario entry class.         */

/*****************************************************************/
/*                                                               */
/*  Class Name : Docket                                          */
/*                                                               */
/*  Description:                                                 */
/*                                                               */
/*  Each Docket must maintain lists of the following object types: */
/*  1.  Notes : This is a generalised object which is used to    */
/*              contain the Note information as follows:         */
/*                  The date and time of entry.                  */
/*                  A message text.                              */
/*              The Note information is divided into the following */
/*              groups: the Operator Note, Control Messages, Clear */
/*              codes, Handover messages, Erased Indications, Cross */
/*              reference messages.                              */
/*                                                               */
/*  2. Alarms : This contains the alarm message with the following */
/*              fields:                                          */
/*              Start date and time, Alarm Category, Alarm       */
/*              description, Alarm urgency, Occurance counter,   */
/*              total duration of the alarm.                     */
/*                                                               */
/*  How it works:                                                */
/*                                                               */
/*  On receipt of a New_Alarm request the LogicalEquipmentGroup  */
/*  object will have to perform the following antics:            */
/*    i.   Ask each Docket in its linked list whether or not they */
/*         can accept the alarm.                                 */
/*    ii.  If the alarm can not be accepted then a new docket must */
/*         be instantiated and given the alarm for further       */
/*         processing. Of course the object must add itself to the */
/*         containment tree once it has a Docket Reference number */
/*         which is generated by DrefGen                         */
/*    iii. Whenever a New_Alarm is processed then that alarm must */
/*         be checked against the GeneralFilter in the case of the */
/*         Docket status been future and against the Docket alarm */
/*         filter in the case of a Current Docket. If a new alarm */
/*         is been added then the alarm must be checked against the */
/*         dockets alarm acceptance time window.                 */
/*                                                               */
/*  Things that Docket should be able to do:                     */
/*                                                               */
/*  During Construction:                                         */
/*                                                               */
/*    i.   Obtain a Docket Reference Number from DrefGenerator. The */
/*         response handler must retry if necessary and once a Dref */
/*         has been obtained then Register the Docket in the MIB, */
/*    ii.  If the Docket is been  created because of a New Alarm then */
/*         create the Alarm entry on the Alarm list and send a  */
/*         Updates to the MMI via the MMIManager. Docket must also */
/*         pass the information to the General Display manager for */
/*         sending to the GeneralDisplay if this docket matches the */
/*         filter settings. The docket must also be checked against */
/*         the Scenario Filter mechanism to see if the presentation */
/*         of the docket must be held off or not. If the case is NOT */
/*         then the docket is made current and presented to the  */
/*         operator.                                             */
/*    iii. If the Docket is been created as a result of the     */
/*         Create_Docket request then set .he Created flag.      */
```

```
121  /*  iv.  The Alarm Summary time information must be constructed and */
122  /*       generated to the HMI indicated.                            */
123  /*                                                                  */
124  /*                                                                  */
125  /*  During Destructions:                                            */
126  /*                                                                  */
127  /*  Release all dynamic memory ( malloc ) and detach all children   */
128  /*  as well as free their resources.                                */
129  /*                                                                  */
130  /*  Other Actions:                                                  */
131  /*                                                                  */
132  /*  i.   Whenever an alarm is updated then notify all displays      */
133  /*       currently active.                                          */
134  /*  ii.  When a Note/ClearCode/Control is added then create a Note  */
135  /*       and add to list.                                           */
136  /*  iii. When Control is performed then add notes as Control        */
137  /*       function progresses.                                       */
138  /*  iv.  When alarms are processed then validate each against the   */
139  /*       Docket alarm filter as well as the general alarms filter   */
140  /*       and if the criteria matches then notify designated         */
141  /*       designated HMI's.                                          */
142  /*  v.   Terminates: Make Docket inactive and notify displays and   */
143  /*       add relative note to list.                                 */
144  /*  vi.  HandOver: Set up Opid list and send a M_SET ( Add ) to the */
145  /*       destination HMI as well as send display updates to already */
146  /*       established displays. Also add relative Note to list.      */
147  /*  vii. HandBack: Update Opid list and send an M_SET ( remove ) to */
148  /*       specific HMI and send updates to all established displays  */
149  /*                                                                  */
150  /*                                                                  */
151  /******************************************************************/
152
153  typedef enum {              /* Enumerated values for elmnt pointers */
154          DOCKET_RESP_CODE = 0,   /* resp_code element number.        */
155          DOCKET_I_S_ID,      /* i_s_id element number.               */
156          DOCKET_LEG_ID,      /* leg_id element number.               */
157          DOCKET_FLAGS,       /* docket flags element number.         */
158          DOCKET_OP_ID,       /* op_id element number.                */
159          DOCKET_OP_TYPE,     /* op_id type .                         */
160          DOCKET_DREF,        /* The current Dref                     */
161          DOCKET_FIRST_SET,   /* First set time                       */
162          DOCKET_LAST_CLEAR,  /* Last clear time                      */
163          DOCKET_TIME,        /* Docket creation time                 */
164          DOCKET_URGENCY,     /* Urgency of docket.                   */
165          DOCKET_LIST_TYPE,   /* List type of docket ( C/F/H )        */
166          DOCKET_STATUS,      /* Object status.                       */
167          DOCKET_F_LOC,       /* Location of fault.                   */
168          DOCKET_PCODE,       /* Progress code of Docket.             */
169          CROSS_REF,          /* Cross Reference Dref number.         */
170          DOCKET_FC_TIME,     /* Final clear time of docket.          */
171          DOCKET_TE_TIME,     /* Docket termination time.             */
172          DOCKET_ACK_TIME,    /* Time of acknowledgement.             */
173          DOCKET_NUM_ATTR
174          }   AttrNo;
175
176  class Docket : public DocketLink    {
177     private:
178
179  /******************************************************************/
180  /* Private data used by the docket level.                         */
181  /******************************************************************/
182
183
184      static Logical first_instance;  /* First instance of this class */
185      static ClassBluePrint
186                          *blue_print;
187      static char *docket_class_name; /* Name of class ( errors )     */
188      void *attr_ptr[DOCKET_NUM_ATTR];/* pointers to actual attributes*/
189
190      unsigned long time_window;      /* The current time window end  */
191
192      unsigned long set_map;          /* Bit map of set categories    */
193      unsigned long clr_map;          /* Bit map of clear categories  */
194      Alarm       *first_alarm;       /* Pointer to first alarm       */
195      Alarm       *last_alarm;        /* Pointer to last alarm        */
196      List        *first_entry;       /* Pointer to first entry.      */
197      List        *last_entry;        /* Pointer to last entry.       */
198      char        *hover_list;        /* Pointer to handiver opids    */
199      Detail      *first_detail;      /* Pointer to first detail      */
200      Detail      *last_detail;       /* Pointer to last detail       */
201      int         alarm_count;        /* Number of alarms in docket   */
202      int         entry_count;        /* Number of entries in docket. */
203      int         detail_count;       /* Number of detail displays    */
204      Leg         *leg_ptr;           /* Pointer Leg object.          */
205      LegSet      *legset_ptr;        /* Pointer to a legset.         */
206      ScEntry     *sc_entry_ptr;      /* Pointer to a scenario entry  */
207      char        *hmi_list;          /* List of destination opids    */
208      DMObject    *alloc_hmi;         /* Allocated HMI pointer        */
209      DMObject    **hmi_ptr_list;     /* Table of associated HMI's    */
210      int         hmi_count;          /* Number of opids              */
211      int         hmi_space;          /* opid storage available.      */
212      long        acc_time;           /* Accumulative time.           */
213      long        acc_count;          /* Accumulative count.          */
214      long        alarm_to;           /* Alarm time out               */
215      long        docket_to;          /* Docket time out.             */
216      TimeoutEntry *docket_timeout;   /* Pointer to time out entry.   */
217      TimeoutEntry *hovers_timeout;   /* Pointer to time out entry.   */
218
219      InvokeID    dref_get;           /* Id number of dref get call   */
220      InvokeID    ccode_create;       /* Id number cr clear code calls*/
221      InvokeID    new_docket;         /* Id number of docket create   */
222      InvokeID    update_docket;      /* Id number of docket update   */
223      InvokeID    delete_docket;      /* Id number of docket delete   */
224      InvokeID    new_alarm;          /* Id number of alarm create    */
225      InvokeID    hover_id;           /* Id number of handover        */
226      InvokeID    scenario_id;        /* Id number of scenario_request*/
227      InvokeID    sth_setid;          /* Id number of scenario_mset   */
228
229      public:
230
231      InvokeID    archive_id;         /* Id number for archive records*/
232
233  /******************************************************************/
234  /* Prototype definitions for docket.                              */
235  /******************************************************************/
236
237  /***  ***************************************************************/
238  /*                                                                  */
239  /*  X           : Docket::Docket                                    */
240  /*                                                                  */
```

```
241  /* Description :                                                    */
242  /*                                                                  */
243  /* Called by the Leg object.                                        */
244  /* Requires the following:                                          */
245  /* 1.  A buffer containing all the concatenated information that    */
246  /*     will represent the ...i object ID as follows:                */
247  /*     Responsibility, Legs , Leg, Equipment, Indication along      */
248  /*     with the class names for each level.                         */
249  /* 2.  A buffer containing the information to be retained by the    */
250  /*     docket which will be sent to the historian and event log     */
251  /*     whenever the alarm or docket has cleared. Note that this     */
252  /*     information will be used to construct the docket summary     */
253  /*     line as well as the first alarm entry.                       */
254  /*     Information required:                                        */
255  /*                                                                  */
256  /*                                                                  */
257  /*******************************************************************/
258  Docket (
259      SomIError       *error,             /* Error return value      */
260      char            *allocation,        /* Dest operator ID        */
261      char            *op_type,           /* Dest operator type      */
262      CMObject        *legset,            /* The LegSet object       */
263      CMObject        *leg,               /* The Leg Object          */
264      CMObject        *equipment,         /* The equipment object    */
265      DILink          *indication,        /* The indication object   */
266      char            *category,          /* category of alarm       */
267      char            *alarm_criteria,    /* The alarm criteria      */
268      short           alarm_urgency,      /* Urgency of this alarm   */
269      long            severity,           /* Severity of alarm       */
270      long            set_time,           /* time of alarm set       */
271      Logical         affected,           /* Alarm is affected       */
272      char            *eqt_id_val,        /* Equipment id            */
273      char            *ind_id_val,        /* Indication id           */
274      Logical         supp_ar );          /* Suppress archiving      */
275
276  /*******************************************************************/
277  /*                                                                 */
278  /* Name        : ~Docket                                           */
279  /*                                                                 */
280  /* Description : The destructor.                                   */
281  /*               Remember to add your own variables to the argument*/
282  /*               list                                              */
283  /*                                                                 */
284  /*******************************************************************/
285  ~Docket ( void );
286
287  /*******************************************************************/
288  /*                                                                 */
289  /* Name        : SetInd                                            */
290  /*                                                                 */
291  /* Description : Sets the specified attributes to the supplied     */
292  /* values. The attributes must first be validated before any       */
293  /* changes are made and an error must be returned if any of the    */
294  /* AVA entries are incorrect.                                      */
295  /*                                                                 */
296  /*******************************************************************/
297  void SetInd(
298      RXMessage   *message,                   /* The receive message */
299      RspBlock    *response,                  /* The response block  */
300      SomIValue   *Attr_Mod_List );           /* Attr to be modified */
301
302  /*******************************************************************/
303  /*                                                                 */
304  /* Name        : ActionInd                                         */
305  /*                                                                 */
306  /* Description : Performs some sort of action.                     */
307  /*                                                                 */
308  /*******************************************************************/
309  void ActionInd(
310      RXMessage   *message,                   /* The receive message */
311      RspBlock    *response,                  /* The response block  */
312      SomIValue   *action_type,               /* Type of action      */
313      SomIValue   *action_info );             /* Action information  */
314
315  /*******************************************************************/
316  /*                                                                 */
317  /* Name        : CreateCnf                                         */
318  /*                                                                 */
319  /* Description :                                                   */
320  /* All creates will return here. Therefore this routine must handle*/
321  /* all of the following scenarios:                                 */
322  /* A Create Dref request:                                          */
323  /* A Create Docket request to the HMI                              */
324  /* A Create detail line if any current detail displays.            */
325  /*                                                                 */
326  /*******************************************************************/
327  void CreateCnf(
328      RXMessage   *rxm );                     /* The receive message */
329
330  /*******************************************************************/
331  /*                                                                 */
332  /* Name        : DeleteCnf                                         */
333  /*                                                                 */
334  /* Description : This routine is called if this object had         */
335  /* requested that an object be deleted via SOMI                    */
336  /*                                                                 */
337  /*******************************************************************/
338  void DeleteCnf(
339      RXMessage   *rxm );                     /* The receive message */
340
341  /*******************************************************************/
342  /*                                                                 */
343  /* Name        : GetCnf                                            */
344  /*                                                                 */
345  /* Description : This routine is called if this object had         */
346  /* requested a get via SOMI.                                       */
347  /*                                                                 */
348  /*******************************************************************/
349  void GetCnf(
350      RXMessage   *rxm );                     /* The receive message */
351
352  /*******************************************************************/
353  /*                                                                 */
354  /* Name        : SetCnf                                            */
355  /*                                                                 */
356  /* Description : This routine is called if this object had         */
357  /* requested a set via SOMI.                                       */
358  /*                                                                 */
359  /*******************************************************************/
360  void SetCnf(
```

```
361     XXMessage   *rxm );                    /* The receive message  */
362
363  /*************************************************************/
364  /*                                                           */
365  /*  Name        : ActionCnf                                  */
366  /*                                                           */
367  /*  Description : This routine is called if this object had  */
368  /*  requested that an action be performed via SOMI           */
369  /*                                                           */
370  /*                                                           */
371  /*                                                           */
372  /*                                                           */
373  /*************************************************************/
374  void ActionCnf(
375      XXMessage   *rxm );                    /* The receive message  */
376
377  /*************************************************************/
378  /*                                                           */
379  /*  Name        : EventReportCnf                             */
380  /*                                                           */
381  /*  Description : This routine is called if this object had  */
382  /*  Sent out an EventReport in confirmed mode.               */
383  /*                                                           */
384  /*************************************************************/
385  void EventReportCnf(
386      XXMessage   *rxm );                    /* The receive message  */
387
388  /*************************************************************/
389  /*                                                           */
390  /*  Name        : RequestDref                                */
391  /*                                                           */
392  /*  Description :                                            */
393  /*  This function will generate the request to fetch the dref*/
394  /*  from the dref generator object.                          */
395  /*                                                           */
396  /*************************************************************/
397  void RequestDref (
398      OH  *local_oh_ptr );                   /* Pointer to the OH    */
399
400  /*************************************************************/
401  /*                                                           */
402  /*  Name        : NewDocket                                  */
403  /*                                                           */
404  /*  Description :                                            */
405  /*  Th   function will construct the information required for the */
406  /*  docket summary line pair and send it to the configured MMI   */
407  /*                                                           */
408  /*************************************************************/
409  SomiError NewDocket(                   /* Returns error on failure */
410      char    *opid,                     /* The opid to send it to   */
411      int     tag );                     /* The opid tag number      */
412
413  /*************************************************************/
414  /*                                                           */
415  /*  Name        : UpdateDocket                               */
416  /*                                                           */
417  /*  Description :                                            */
418  /*  This function will send the information required for updating */
419  /*  the docket summary line and send it to the configured MMI.   */
420  /*                                                           */
421  /*************************************************************/
422  SomiError UpdateDocket(                /* Returns error on failure */
423      void );
424
425  /*************************************************************/
426  /*                                                           */
427  /*  Name        : DeleteDocket                               */
428  /*                                                           */
429  /*  Description :                                            */
430  /*  This function will send a request  to the MMI to remove the  */
431  /*  docket from its summary list.                            */
432  /*                                                           */
433  /*************************************************************/
434  SomiError DeleteDocket(                /* Returns error on failure */
435      int     tag );                     /* MMI Entry tag            */
436
437  /*************************************************************/
438  /*                                                           */
439  /*  Name        : ClearAlarm                                 */
440  /*                                                           */
441  /*  Description :                                            */
442  /*  This function is used to inform docket of an alarm going clear.*/
443  /*                                                           */
444  /*************************************************************/
445  Logical ClearAlarm(                    /* Return true if problems  */
446      char            *eqt_id_val,       /* Equipment id             */
447      char            *ind_id_val,       /* Indication id            */
448      char            *category,         /* The alarm category       */
449      char            *criteria,         /* The alarm criteria       */
450      long            clear_time,        /* Clear time of alarm      */
451      Logical         affected,          /* Alarm is affected        */
452      Logical         supp_ar);          /* Alarm must be archived   */
453
454  /*************************************************************/
455  /*                                                           */
456  /*  Name        : SetAlarm                                   */
457  /*                                                           */
458  /*  Description :                                            */
459  /*  This function is invoked by the Leg class to either update an */
460  /*  existing alarm or add the alarm to the docket. If the alarm  */
461  /*  belongs to the docket but cannot be accepted the a -1 will be */
462  /*  returned to Leg. If the alarm is accepted then a 0 will be   */
463  /*  returned, If the alarm cannot be accepted then 1 will be     */
464  /*  returned.                                                */
465  /*                                                           */
466  /*************************************************************/
467  Logical SetAlarm(                      /* False if alrm not valid  */
468      CMObject        *equipment,        /* Pointer to Equipment     */
469      DLLink          *indication,       /* Pointer to Indication    */
470      char            *category,         /* The category of the alrm */
471      char            *criteria,         /* Criteria of alarm        */
472      short           urgency,           /* Urgency of alarm         */
473      long            severity,          /* Severity of the alarm    */
474      long            set_time,          /* Time of alarm            */
475      Logical         affected,          /* Alarm is affected        */
476      char            *eqt_id_val,       /* Equipment id             */
477      char            *ind_id_val,       /* Indication id            */
478      Logical         supp_ar);          /* Suppress archiving       */
479
480  /*************************************************************/
```

REFERENCES

1 Ahmed S., Wong A., Sriram D., Logcher R., Object-oriented Database Management Systems for Engineering: A Comparison, *Journal of object-oriented Programming*, vol.5, no.3, p27-43, June 1992.

2 Anderson B., Gossain S., *Hierarchy Evolution and the Software Lifecycle*, (publication source unknown), Dep. Electronic Systems Engineering, University of Essex, England.

3 Arango G., *Self-explained Toolboxes: A Practical Approach to Re-usability*, (source unknown)

4 As quoted in Sommerville I., *Software Engineering*, 3rd edition, Addison-Wesley, p546, 1989.

5 Barnes B.H., Bollinger T.B., Making Re-use Cost-Effective, *IEEE Software*, p14-24, Jan. 1991.

6 Basili V.R., Viewing Maintenance as Reuse-Oriented Software Development, *IEEE Software*, p19-25, January 1990.

7 Basset P. G., Frame-based Software Engineering, *IEEE Software*, p9-16, Jul. 1987.

8 Beck K., Cunningham W., A Laboratory for Teaching Object-oriented Thinking, *Proceedings of OOPSLA*, 1989.

9 Beck K., Think Like an Object, *UNIX Review*, vol.9, no.10, p39-43, November 1991.

10 Biggerstaff T., Richter C., Reusability Framework, Assessment, and Directions, *IEEE Software*, p41-49, March 1987.

11 Boehm B.W., Software Engineering, *IEEE Transactions Computers*, p1226-1241, 1976.

12 Boehm B.W., Software Risk Management: Principles and Practices, *IEEE Software*, p32-41, January 1991.

13 Booch G., *Object Oriented Design with Applications*, Benjamin/Cummings Publishing Company, 1st ed., p187-195, p472-511, 1991.

14 Borkin S.A., *Data Models: A Semantic Approach for Database Systems*, The MIT Press, 1980.

15 Brooks F., No Silver Bullet: Essence and Accidents of Software Engineering, *IEEE Computer*, Apr. 1987.

16 CCITT Study Group XV, *Draft Recommendation G.773 - Protocol Suites for Q Interfaces for Management of Transmission Systems*, Geneva Meeting 16-27 July 1990, CCITT, August 1990.

17 Chidamber S.R., Kemerer C.F., Towards a Metric Suite for Object Oriented Design, *OOPSLA '91*, p197-211, 1991.

18 Coad P., Yourdon E., *Object Oriented Analysis*, Prentice-Hall, Engelwood Cliffs, 1990.

19 Cox B.J., The Economics of Software Re-use, *Panel discussion: OOPSLA '91*, p264-270, 1991.

20 Cox B., There is a Silver Bullet, *BYTE*, p209-218, Oct. 1990.

21 DeMarco T., *Structured Analysis and System Specification*, Yourdon Press, Engelwood Cliffs, 1979.

22 Du Plessis A., Van der Walt E., *Modelling the Software Development Process, IFIP WG8.1 Working Conference on Information Systems Concepts (Alexandria, Egypt)*, p1-26, April 1992.

23 Duff C., Howard B., Migration Patterns, *BYTE*, p223-232, Oct. 1990.

24 Fischer G., Cognitive View of Reuse and Redesign, *IEEE Software*, p60-72, July 1987.

25 Gane C., Sarson T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Engelwood Cliffs, 1979.

26 Gangopadhyay D, Helm R., *A Model Driven Approach for the Reuse of Classes from Domain-specific Object-oriented Classes*, Technical Report RC14510, IBM Research Division, March 1989.

27 Gibbs S., Tsichritzis D., Casais E., Class Management for Software Communities, *Communications of the ACM*, vol. 33, no. 9, p90-103, September 1990.

28 Gibson E., Objects - Born and Bred, *BYTE*, p245-254, Oct. 1990.

29 Goldberg A., Rubin K.S., Taming Object-oriented Technology, *Computer Language*, p34-45, Oct. 1990.

30 Guimaraes N., Building Generic User Interface Tools: an Experience with Multiple Inheritance, *OOPSLA '91*, p89-95, 1991.

31 Hampel T.L., *Understanding Unix Industry Consortiums and Standards Groups*, Marketing Communication, WYSE Technology Inc., August 26 1986.

32 Hayes F., How POSIX is redefining UNIX, *UnixWorld*, December 1990.

33 Held G., *Understanding Data Communications*, John Wiley and Sons, 1991.

34 Helm R., Maarek Y.S., Integrating Information Retrieval and Domain Specific Approaches for Browsing and Retrieval in Object-Oriented Class Libraries, *OOPSLA '91*, p47-61, 1991.

35 Henderson-Sellers B., Edwards J.M., The Object-oriented Systems Life Cycle, *Communications of the ACM*, vol.33, no.9 ,p142-159, Sep. 1990.

36 Jacobson I., Industrial Development of Software with an Object-oriented Technique, *Journal of Object-oriented Programming*, p30-40, Mar/Apr 1991.

37 Johnson R.E., Foote B., Designing Re-usable Classes, *Journal of Object-oriented Programming*, p22-35, June/July 1988.

38 Jrad A.M., Lin C., Rebman T.M., Object-oriented Software in Real-time Systems, *IEEE Software*, vol.4, no.6, p77-89, Oct. 1989.

39 Laranjeira L.A., Software Size Estimation of Object-Oriented Systems, *IEEE Transactions on Software Engineering*, vol.16, no.5, p510-521, May 1990.

40 Lehman M., Programs, Life Cycles, and Laws of Software Evolution, *Proceedings of the IEEE*, vol.68, no.9 , p1060-1076, Sep. 1980.

41 Lewis A.J., Henry S.M., Kafura D.G., An Empirical Study of the Object-oriented Paradigm and Software Re-use, *OOPSLA '91*, p184-196, 1991.

42 Lewis A.J., Henry S.M., Kafura D.G., Schulman R.S., On the Relationship between the Object-oriented Paradigm and Software Re-use: an Empirical Investigation, *Journal of object-oriented Programming*, vol.4, no.4, p35-41, July/August 1992.

43 Macro A., Buxton J., *The Craft of Software Engineering*, Addison-Wesley Publishing Company, 1st ed., p1-44, 1987.

44 Meyer B., Lessons from the Design of the Eiffel Libraries, *Communications of the ACM*, vol. 33, no. 9, p69-88, September 1990.

45 Meyer B., Object-oriented Software Construction, *Journal of Object-oriented Programming*, Prentice-Hall, Hemel Hempstead, 1988.

46 Meyer B., Re-usability: The Case for Object-oriented Design, *IEEE Software*, p50-64, March 1987.

47 Meyers W., Interview with Wilma Osborne, *IEEE Software*, vol.5, no.3, p104-105, 1988.

48 Pliskin N., Balalia I., Kenigshtein I., The Knowledge Contribution of Engineers to Software Development: A Case STudy, *IEEE Transactions on Engineering Management*, vol.38, no.4 ,p344-347, Nov. 1991.

49 Prieto Diaz R., Freeman P., Classifying Software for Reusability, *IEEE Software*, vol.4, no.1, p6-16, January 1987.

50 Ross T.D., Goodenough J.B., Irvine C.A., Software Engineering: Process, Principles and Goals, *Computer*, p54-64, May 1975.

51 Royce W.W., Managing the Development of Large Software Systems: Concepts and Techniques, *Proceedings, WESCON*, 1970.

52 Stone C.M., Hentchel D., Database Wars Revisited, *BYTE*, vol., no., p223-242, October 1990.

53 Ward P.T., Mellor S.J., *Structured Development for Real-Time Systems (Volumes 1-4)*, Yourdon Press, Engelwood Cliffs, 1985.

54 Wegner P., Perspectives on Object Oriented Programming, *Private communication*, 1986.

55 Wirfs-Brock R.J., Johnson R.E., Surveying Current Research in Object-Oriented Design, *Communications of the ACM*, vol.33, no.9, p105-123, September 1990.

56 Wirfs-Brock R.J., Wilkerson B., Object Oriented Design: a Responsibility Driven Approach, *Proceedings of OOPSLA*, vol., no., p71-75, 1989.

57 Yourdon E.N., *Managing the System Life-cycle: a Software Development Methodology Overview*, Yourdon Press, 1982.

58 Yourdon E., Auld Lang Syne, *BYTE*, p257-262, Oct. 1990.

BIBLIOGRAPHY

Further reading on object-orientation:

1 Booch G., *Object Oriented Design with Applications,* The Benjamin/Cummings Publishing Company, 1991.

Further reading on Open Systems and Inter-networking standards and protocols:

1 Black U.D., *Data Communications and Distributed Networks*, Prentice-Hall International, 2nd Edition,1987.

2 Comer D.E., Stevens D.L., *Internetworking with TCP/IP (Volumes 2),* Prentice-Hall International, 1991.

3 Comer D.E., *Internetworking with TCP/IP (Volumes 1),* Prentice-Hall International, 2nd Edition, 1991.

4 Held G., *Understanding Data Communications*, John Wiley and Sons, 1991.

5 Network Working Group, *The Common Management Information Services and Protocol over TCP/IP (CMOT)*, April 1989.

6 OSI/Information Technology, *Common Management Information Protocol Specification,* Final text of DIS 9596, January 1990.

7 OSI/Information Technology, *Common Management Information Service Definition*, Final text if DIS 9595, January 1990.

8 OSI/Information Technology, *Structure of Management Information - Part 1: Management Information Model*, ISO/IEC Proposal 10165-1, June 1989.

9 OSI/Information Technology, *Structure of Management Information - Part 2: Definition of Management Information,* ISO/IEC Proposal 10165-2, December 1989.

10 OSI/Information Technology, *Structure of Management Information - Part 3: Definition of Management Attributes*, ISO/IEC Proposal 10165-3, September 1989.

11 OSI/Information Technology, *Specification for Abstract Syntax Notation One (ASN.1)*, Final text for DIS 8824, April 1990.

12 OSI/Network Management Forum, *Application Services,* Issue 1, June 1989.

13 OSI/Network Management Forum, *Protocol Specification*, Issue 1, January 1989.

14 Van Norman H.J., *LAN-WAN Internetworking*, Technical Report, Faulkner, 1991.