

Object Oriented Implementation & Design

A Thesis by: GERARD MC CLOSKEY BSc.
Supervisors: DR. TONY MOYNIHAN PhD.
MR. RENAAT VERBRUGGEN M.M.I.

Submitted to
DUBLIN CITY UNIVERSITY
COMPUTER APPLICATIONS
for the degree of
Master of Science
August 1989

Declaration No portion of this work has been submitted in support of an application for another degree or qualification in the Dublin City University or any other University or Institute of Learning.

Acknowledgement

I would like to express my appreciation to Tony Moynihan, Renaat Verbruggen and Martin Doherty for all their help and encouragement while working at Dublin City University. To the Fitzsimons family for their patience and encouragement while writing this thesis,

and most of all to my Mum and Dad for their support while completing these studies.

OBJECT ORIENTED IMPLEMENTATION & DESIGN

Abstract

Author: Gerard Mc Closkey

As the power and speed of computers increase so too are the number of applications to which they can be applied. Unfortunately software development is not performed using the approach taken by hardware engineers where development proceeds from previous innovations. This has caused software applications to be developed at a much slower rate than their hardware counterparts.

Traditional software development usually starts from scratch, despite the commonality that exists between various applications. The Object Oriented Approach is different from the conventional approach in that it encourages the reusing and extending of existing software modules in a manner similar to hardware development. The characteristics of the approach (i.e. inheritance, encapsulation and dynamic binding) signals the need for new methodologies and new implementation techniques.

Using the Object Oriented language Objective-C for the creation of an "Integrated Management Process Workbench" various issues associated with this approach are discussed in this text. The "Integrated Management Process Workbench" was an Esprit project in which Dublin City University was involved along with a number of partners from sites throughout Europe. The Calendar and Risk Analysis tools developed in Dublin for the workbench provided the development application on which the discussion is centered.

Contents

1	Object Oriented Terminology	1
1.1	Introduction	1
1.2	Thesis Overview	2
1.2.1	Integrated Management Process Workbench (IMPW) . . .	3
1.3	Birth of Object Oriented languages	4
1.4	What makes a language Object Oriented ?	5
1.4.1	Encapsulation	6
1.4.2	Inheritance	9
1.4.3	Other Object Oriented characteristics	11
1.5	Summary	12
2	Objective-C	13
2.1	Introduction	13
2.2	Objective-C data types	14
2.2.1	Class and Instance Objects	14
2.3	Messages	15

2.4	Methods	17
2.5	Class Definition Files	18
2.6	Self and Super	21
2.7	Objective-C Inheritance Library	22
2.7.1	Foundation Classes	23
2.7.2	Collection Classes	25
2.8	Tying it all together	26
2.9	Objective-C Compiler	28
2.10	Summary	30
3	Risk Analysis Tool	31
3.1	Introduction	31
3.2	Risk Tool Overview	32
3.2.1	Tool Input	32
3.2.2	Tool Output	32
3.3	Important Classes	33
3.3.1	The Risk Class	33
3.3.2	The Rule Class	34
3.3.3	The Text Class	38
3.3.4	The Measure Class	39
3.4	Types of RiskTool Users	40
3.4.1	SuperUser Functionality	41

3.4.2	The Project Manager	47
3.5	Summary	52
4	Calendar Tool	54
4.1	Introduction	54
4.2	Calendar Overview	55
4.3	Calendar Presentation	56
4.4	Date Class	58
4.4.1	Storing the Date	59
4.4.2	The Date Collection	61
4.4.3	Presentation of Dates	62
4.5	CALoad class	63
4.6	Task Class	64
4.7	CalAutomata and State classes	65
4.8	Operation modes	65
4.8.1	Open and Interval Modes	67
4.8.2	Task Mode	68
4.9	Event Details	68
4.10	Summary	70
5	The Workbench Interfaces	71
5.1	Introduction	71
5.2	Automata	72

5.3	Graphic Compatibility	76
5.4	Display Constraints	78
5.4.1	Window Displays	79
5.4.2	Composition of riskdriver window	83
5.4.3	Top row details	84
5.5	Tool Interaction	86
5.5.1	Using the mouse	87
5.5.2	Textual input	88
5.6	Calendar View	89
5.7	Summary	91
6	Objective-C Traps and Pitfalls	92
6.1	Introduction	92
6.2	Objective-C Economics	93
6.2.1	Memory Costs	93
6.2.2	Code Size	96
6.2.3	Binary Size	98
6.2.4	Messaging Overhead	98
6.3	Error Clinic	100
6.3.1	Class Definition Troubles	101
6.3.2	Erroneous Methods	102
6.3.3	Main Module Structure	103

6.3.4	Printing Errors and Error Messages	104
6.3.5	Collection Errors	105
6.4	Garbage Collection	106
6.5	Inheritance	107
6.6	Summary	107
7	Object Oriented Design	109
7.1	Introduction	109
7.2	Booch Model	110
7.3	Hierarchical Object Oriented design(HOOD)	112
7.4	Block Design	115
7.5	Object Oriented Structured Design	116
7.6	Learn by Example	118
7.7	Methodology for workbench tools	118
7.8	Designing Resusable Classes	121
7.9	Summary	123
8	Future Directions	124
8.1	Introduction	124
8.2	Future Enhancements to Objective-C	125
8.3	Future advancements to the Workbench	127
8.4	User Interfaces	128
8.5	Conventional Systems	129

8.6	Object Oriented languages	130
8.6.1	Naming Conventions	131
8.6.2	Class Library Structures	131
8.6.3	Design	131
8.7	Summary	134

List of Figures

1.1	Integrated Management Process Workbench	4
1.2	Encapsulated Object	7
1.3	Bank Objects	8
1.4	An object has two parts	9
1.5	Inheritance Mechanism	10
2.1	Object Declaration	14
2.2	Message Syntax	15
2.3	Message Glossary	16
2.4	Objective-C Message Conventions	16
2.5	Objective-C Statements	17
2.6	Room Class Definition File	20
2.7	Method Structure	21
2.8	Objective-C Hierarchical Inheritance Structure	24
2.9	Objective-C in Memory	27
3.1	Instance Variables for the Risk Class	34

3.2	Instance Variables for the Rule Class	35
3.3	Storing Rule Objects	36
3.4	Sample Rules	37
3.5	Rule and Txt objects.	39
3.6	Initialisation of Riskdriver Instance Variables	42
3.7	Riskdriver Instance	42
3.8	Deleting Riskdrivers	44
3.9	Getting risk values from the database	48
3.10	Rule operands for Rule Number five	49
3.11	Operand and Operator queues	50
3.12	Rule Tree	52
4.1	Calendar Tool Classes	56
4.2	Date Classes Instance Variables	59
4.3	Date Conversion Formula	60
4.4	Task Class Instance Variables	65
4.5	Calendar Automata	66
4.6	Calendar View	67
4.7	Calendar Interrogation	69
5.1	Generic execute method	73
5.2	Risk Automata	74
5.3	Automata within an automata	75

5.4	Instance Variables for RiskAutomata	76
5.5	followingState method	77
5.6	Ideal environment for graphic packages	78
5.7	Realistic graphic package environment	79
5.8	Window layout	80
5.9	Tree presentation	81
5.10	Final window layout	82
5.11	Presentation of Tool and Project names	85
5.12	Initial Risk Window	87
5.13	Amend Riskdriver Window	88
5.14	Calendar Interface	90
6.1	Object Overhead	94
6.2	Objective-C required in memory	95
6.3	Objective-C productivity	97
6.4	Message Overhead	99
6.5	Objective-C performance	100
6.6	Saving messaging time	101
6.7	Method Syntax	103
7.1	Risk Booch-gram	112
7.2	HOOD diagrams	113
7.3	Message passing with HOOD	114

7.4	HOOD inheritance	115
7.5	Object Oriented Structured Design of the Calendar	117
7.6	Object Oriented Structured Design for inheritance	120
8.1	Ideal Objective-C environment	126
8.2	Possible Window Hierarchy	129
8.3	Object Database Hierarchy	133

Chapter 1

Object Oriented Terminology

1.1 Introduction

During the past couple of decades computer professionals have seen a number of advances in their industry. Most of these changes have been directed towards the area of computer hardware. Gradual refinements with hardware technology have meant quantum jumps in the power and facilities offered by processors, whose range of uses may vary from controlling water temperature in a washing machine to controlling an aircraft's flight. The advancements in software over the same time period measure poorly in comparison with these hardware advances.

One of the most contrasting differences between software and hardware has been the way in which project development is performed. When designing electronic circuitry for computer hardware, large portions of the design are made up from circuits previously designed and tested. Innovations in computer hardware use a certain amount of inherited work and design from previous accomplishments in this area.

With computer software a different approach is used, computer scientists usually start writing code from scratch. The professional computer person will have a number of design methodologies in his tool kit plus a number of library routines to help develop a computer system. However no attempt will be made to inherit tried and tested software from another programmer for application development. Software people instead will spend time and care developing algorithms and tools similar to what many other people in their profession have done and continually repeat. Computer systems developed today share a large

amount of commonality with thousands of other packages developed during the past decade. Algorithms using sorts, linked lists and binary trees continually recur in computer systems. For the software professional rewriting and testing these algorithms, their work may seem analogous to reinventing the wheel.

Object Oriented languages introduce an approach which unlike conventional languages, encourages reuse of code so that factors associated with hardware such as high quality, correct, robust, extendible, reusable and compatible units can be applied to software[Mey 87]. This thesis looks at the characteristics which make a language Object Oriented and then investigates how it is applied to the areas of implementation and design. In practice the construction of code should not be considered before analysis and design has taken place. However as if to underline the difference between Object Oriented and conventional approaches the thesis starts with a discussion on implementation before design. This is done to introduce the concepts of Object Oriented languages to the naive reader as early as possible.

1.2 Thesis Overview

The first chapter describes the characteristics which make a language Object Oriented and the "Integrated Management Process Workbench" which was constructed using Object Oriented methodologies. The Object Oriented language used for the development of this workbench is Objective-C, which is a hybrid language consisting of C and Smalltalk. The second chapter in the thesis, explains some of the basic statements and syntax required for beginning development in this environment. The following two chapters describe workbench tools (i.e. Risk and Calendar) which have been constructed using these principles giving additional examples of Objective-C statements whenever appropriate.

Chapter five talks about the graphic interface used for the creation of these tools and how it interacted with the Objective-C environment. Chapter six moves away from the creation of the IMPW using Objective-C, towards studying the economics of the approach. This chapter investigates what additional price has to be paid for an Object Oriented environment such as Objective-C, in terms of machine space, code size, development time, operation efficiency etc.

The last two chapters concentrate on design, discussing the various approaches to be adopted when using Object Oriented environments, and the one that was selected for the development of the IMPW. The thesis ends by taking a brief look into the future role of Object Oriented languages.

Throughout this text explanations relating to Object Oriented programming, the Objective-C environment, and the IMPW were kept as simple and non technical as possible. Hopefully this will encourage as many readers as possible who are interested in Object Oriented technology and workbench environments, to read this text. While aiming mainly at the newcomer to the approach, it is hoped that some of the text will be of use to people already familiar with Object Oriented programming and design, pointing them towards new ideas for this relatively new technology.

1.2.1 Integrated Management Process Workbench (IMPW)

My interest in Object Oriented programming and design was triggered by involvement in the development of the Integrated Management Process Workbench (IMPW) a project sponsored by the European Esprit(European Strategic Research Program Information Technology) committee. Throughout this text there will be a number of references to the IMPW, so its explanation is given now so that the reader may understand its architecture and where the tools created using Objective-C, fit in this environment. The overall function of the IMPW was to develop a workbench which would help project managers in planning software projects. The IMPW permitted integrated planning of tasks, staff allocation, quality management and risk assessment. It also provides extensive monitoring facilities to enable the project manager to control a project. The size of this project and the geographical distribution of software partners contributed to the decision for adopting an Object Oriented approach.

The architecture of the workbench consists of three main components which can be thought of as three objects as listed below.

Information System (IS) consists of a relational database system holding the details of the project and the software engineering characteristics. It is linked to a Prolog system containing knowledge about the methods and tools of software management and a document storage system based on UNIX filestore. By using a database containing records of previous projects, as well as the current project coupled to an inference engine, the workbench can provide decision support and expert system facilities to managers.

Manager Workbench Interface (MWI) comprised of the I/O manager responsible for all interactions with the manager and the workbench controller responsible for tool control.

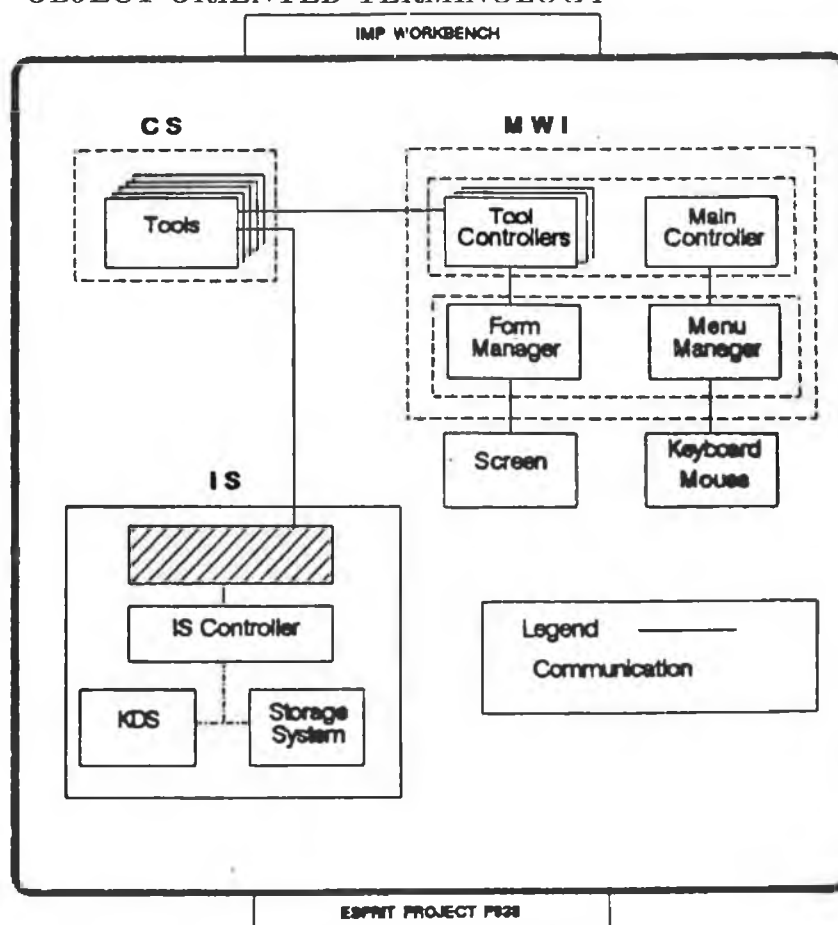


Figure 1.1: Integrated Management Process Workbench

Computational System (CS) includes a range of tools and utilities, such as the Risk Analysis, Estimation and Resource monitoring tools.

A translator exists with the IS so that when components are communicating with the IS their messages are first translated into the formal language of the IS so that they can be acted on by the IS controller.

1.3 Birth of Object Oriented languages

The concept of Object Oriented languages was first developed more than twenty years ago. Simula67 developed in Oslo 1967[Dah 66] was the first Object Oriented language. However despite grouping software modules into objects the language failed to receive widespread recognition. It was Smalltalk[Gol 83] which built on the concepts of Simula67, adding the inheritance property that proved, the innovation, which for many people separates conventional third generation languages

from Object Oriented languages, which made the real breakthrough.

But why has it taken the computer industry almost twenty years to bring the concept of Object Oriented languages to prominence? The reasons for Object Oriented languages becoming popular can be attributed to a number of factors.

- The storage space provided by computer chips has increased while the price has decreased.
- Increased popularity of iconic interfaces introduced by the Macintosh, where everything is displayed as iconic objects.
- Continual backlog of software projects which still haunts many DP departments, a new approach is needed where the code is more reusable and reliable.

Object Oriented languages introduce a new view for software development, one where more emphasis is placed on reusable and hence more reliable software. Unlike more traditional programming methods that are based on concepts such as data flow or mathematical logic Object Oriented programming directly models real world entities. During the past decade OBJECT ORIENTED programming has become a popular buzzword in the computer industry. As this popularity has increased, so too has the number of object oriented languages in the commercial marketplace. The list of Object Oriented languages goes from Simula and LISP to Smalltalk-80, C++, Object Pascal, Objective-C, Eiffel etc, each vary in syntax and in what mechanisms they offer, but they all claim that they are object oriented.

1.4 What makes a language Object Oriented ?

Because commercial object oriented languages are still in their infancy no international standards have been set to determine what an object oriented language should be composed off. Conversations as to what features a language must have, to be categorised as Object Oriented will continue for some years to come. Similar problems existed when database technology became available. Some database

people still argue whether a database has the correct characteristics which make it relational or non relational.

Differentiation between what does and what does not make languages Object Oriented will not be discussed here. Instead this chapter concentrates on the main characteristics that are desirable for an Object Oriented language to have. We will start by defining the object.

Object Oriented programming is as the name suggests programming in objects, but what exactly is an object?. With Object Oriented programming an object may be anything the programmer wishes, surrounded by a number of related procedures.

Object: *Defⁿ* some data, a group of operations on that data,
 and a mechanism for selecting an operation given
 a command.

Conceptually an object can be thought of as a machine capable of performing some predefined actions in response to messages.

1.4.1 Encapsulation

The main characteristic of an Object Oriented language and one which is at the heart of the whole approach is data encapsulation which is unlike conventional programming languages where the data and procedures are taken as two separate components. Object Oriented systems combine the data and the procedures. This encapsulated module is known as the object. With the Object Oriented approach the data which is surrounded by a number of procedures is private to the object. The data encapsulated may only be manipulated by one of these surrounding procedures. The model below in figure 1.2 shows an object -**myBankAccount**. The center contains the various data associated with - myBankAccount i.e. name, address, tele No, sex, account number etc. The procedures surrounding the data are the operations which may be performed by that object.

The procedures form a wall of code around the objects data, whereby all access is handled by one of the object's procedures built exactly for that function. For example when bank interest is added to myBankAccount, the interest procedure will handle this operation. Unlike conventional languages a significant change has been made in the role played by data. Using conventional languages the user would have been responsible for applying the correct data and data types

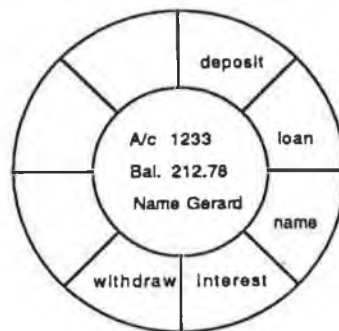


Figure 1.2: Encapsulated Object

eg. the correct amount and the correct format must be specified before the bank interest is calculated. With the Object Oriented approach the user only has to specify the procedure interest leaving the rest to the Object Oriented mechanism.

This simple example underlines one of the main advantages of Object Oriented Programming, the emphasis on the user to supply the correct data in the correct format has been removed. The responsibility has shifted to the supplier of the procedure to supply a correct, efficient, and reliable procedure. Not only does this reduce user input but more importantly it reduces the number of things the user needs to know about the system. In the example above the user is only concerned that the correct interest is added to his savings. How this is performed is not important, so long as it performs its function correctly when asked to. The object knows its own private data, it only has to call the interest procedure which is an operation that can only be performed on the data inside the object. Another benefit of this technique, is that changes can be made inside the encapsulated object without affecting applications using it. The user of an object only sees the services provided by an object, not how the actual service is implemented. The operation for calculating interest can be changed without affecting the user of this procedure, as long as the external view of the procedure remains the same.

Classes and Instances When the bank system above needs other accounts, duplicating the procedures and surrounding them with data for each new bank account as in figure 1.3 would be very wasteful. Copying the code for each new account and changing all the account objects when a change is made to one of

the procedures, say the interest routine again, would be regarded as tedious to say the least. There would also be the problem of space due to copying similar procedures. Having a system full of duplicate objects is not only wasteful causing storage problems in large systems, where many objects are active simultaneously but it also increases maintenance when a change has to be made to a procedure. If the interest rate has to be changed in the example above it would have to be changed for all objects which may exist.



Figure 1.3: Bank Objects

Most Object Oriented languages, Objective-C included make a distinction between the description of the object and the object itself[Rob 81]. Many similar objects are described by some general description. This description of an object is called a class (or shared part), since a class can describe a whole set of related objects. Each object described by a class is called an instance (or private part) of that class. The diagram in figure 1.4 shows this more economical representation adopted by most Object Oriented languages.

In programming language research, information hiding (encapsulation) has been the guiding principle in the development of abstract data systems found in languages such as Alphard, CLU and Ada. While many of these systems lack the concepts of inheritance, they constitute an important class of system called "Object Based Systems"[Weg 88].

myAccount

hisAccount

herAccount

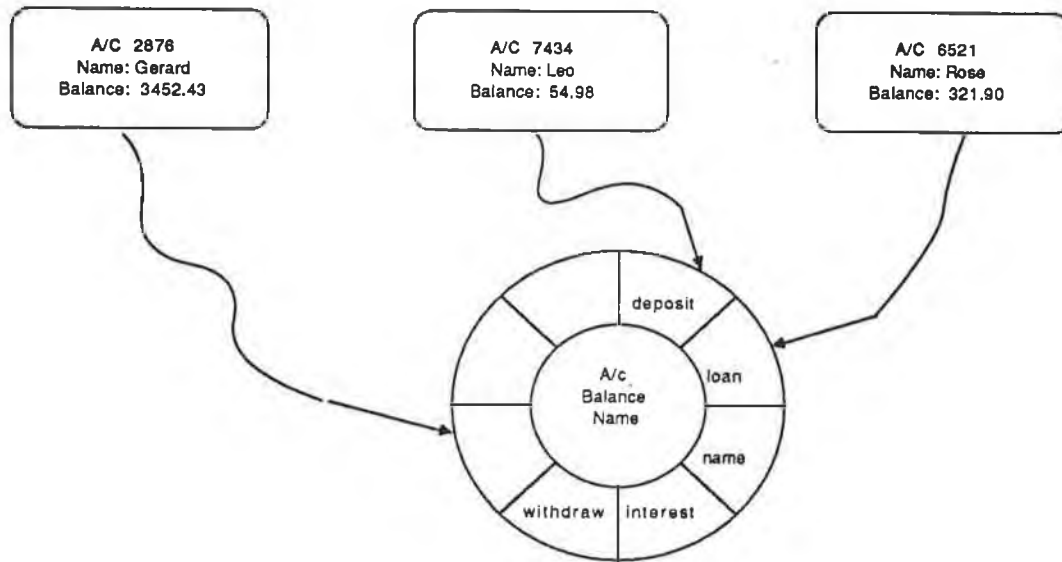


Figure 1.4: An object has two parts

1.4.2 Inheritance

Inheritance is the major feature distinguishing Object Oriented programming systems from conventional programming systems and Object Oriented languages from object based systems. Inheritance increases code reuse by allowing the programmer to inherit code from generic classes[Hal 88]. This feature means that the programmer does not have to redesign each new class of object explicitly. Inheritance permits generic operations to be inherited in the way we inherit knowledge.

The tree in figure 1.5 shows a subset of the animal hierarchical structure. From this diagram we know that Peter and Leo are both men and will therefore inherit all the characteristics which are common to men. Details of how many legs, arms, bones in the body is determined automatically from the parent nodes in the tree. The inheritance of objects using Object Oriented languages follow a similar pattern, the knowledge inherited is the data and procedures from parent objects. By inheriting the procedures and data types common to objects not only is space saved, but also maintenance is reduced. Procedures common to many objects can be maintained in isolation leaving the calling object unaffected.

This makes inheritance a powerful tool for building systems, by organising objects into related groups and using previously defined procedures. Without inheritance objects would be freestanding units, which would have to be developed from scratch. Any consistency between objects inside computer systems would

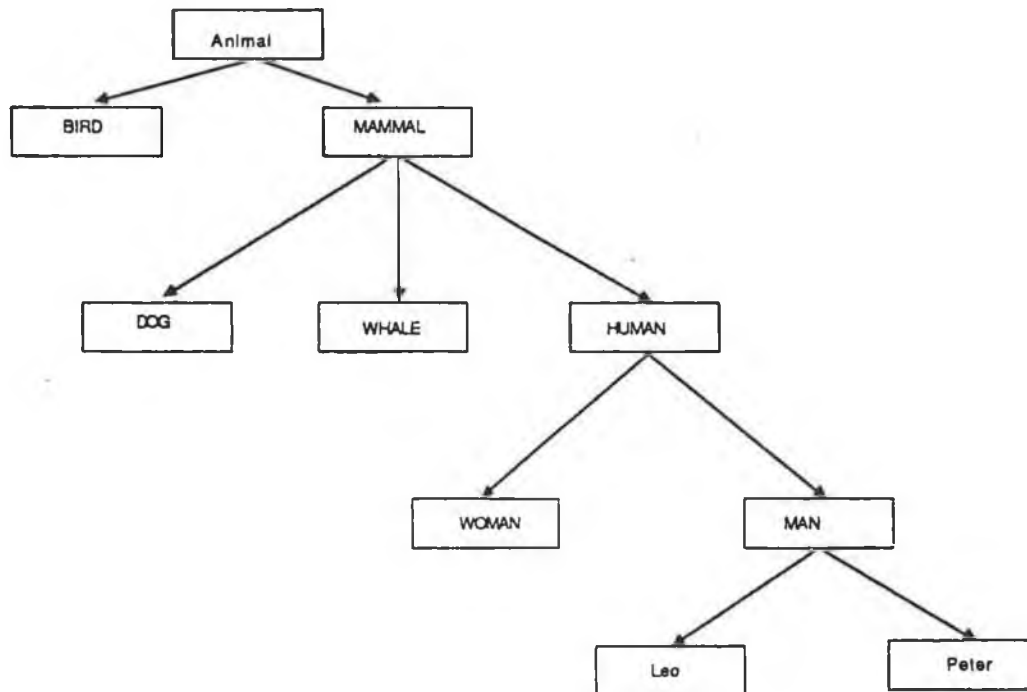


Figure 1.5: Inheritance Mechanism

only be achieved through discipline by the programmer.

Object Oriented languages which permit inheritance may simulate similar hierarchical trees into computer systems. When an object inherits similar knowledge i.e. procedures and data, the work required for amending and extending a computer system is greatly reduced. The fact that code already exists in inherited objects substantially reduces the amount of code to be developed by the programmer. The power of encapsulation and inheritance is optimised by creating carefully designed objects, where changes may be made to objects causing little or no effect on other object modules.

1.4.3 Other Object Oriented characteristics

Encapsulation and inheritance are the two most desirable properties required by Object Oriented languages. Authors from various books, journals, and papers on this topic may state other requirements. Their reasons are often related to certain features offered by the Object Oriented environment in which they are working. Some of these features are mentioned briefly below.

Dynamic Binding With conventional languages the operations related to a routine are assigned at compile time. However for systems to have knowledge about all possible operations during its life-cycle is an almost impossible task. Dynamic binding helps by postponing the decision of what operation to invoke until run time.

Conventional languages such as C, Pascal etc, offer some dynamic binding using case statements. Unfortunately the addition of further operations cause modification of existing code. Dynamic or late binding as it is sometimes called, does not suffer from this problem in Object Oriented languages, thanks to encapsulation and inheritance.

Critics of dynamic binding argue that the approach affects the speed at which applications run, while proponents of dynamic binding argue that its good for rapid prototyping and end user programming. Certainly there is a price to be paid for not specifying the operations until run time. However this degradation in performance, is outweighed by the benefits gained in flexibility where the user may postpone decisions on types and structure which are not relevant until run time.

The operations performed by objects are implemented using procedures (known as messages in Object Oriented technology) similar to conventional procedural calls. However instead of the procedure name identifying the code for computation, in Object Oriented languages this action is performed by the object. The operation performed by a print message for example, would depend on the object that receives the print message. If it was a rectangular object then it would print a rectangle, a circular object would print a circle and so on. The ability of different objects to perform different operations on identical message calls, is called **polymorphism**.

Multiple Inheritance The inheritance mechanism discussed throughout this text is related to single inheritance. Some languages such as C++, Eiffel, CLOS allow an object to have more than one parent. Such systems are said to provide multiple inheritance. Systems which allow multiple inheritance increase code

sharing by making it possible to combine descriptions from several classes. For it is possible to have an object such as “lorry”, which has both vehicle and toy as parent classes. Most languages which support multiple inheritance use some type of precedence relationship to indicate which class should be dominant[Ste 86]. Other languages such as Smalltalk-80 take the stand that no simple precedence relationship for multiple inheritance will work, leaving the responsibility as to which class should have precedence to the user, this approach can have difficulties solving ambiguities between objects.

1.5 Summary

This chapter has given the reader an introduction to Object Oriented programming languages. No mention of specific language syntax was made, only Object Oriented concepts have been described. Understanding the principles of encapsulation (i.e. a seamless module whose data can only be manipulated by one of its surrounding procedures), inheritance (i.e. the ability to build onto already known and valid information) dynamic binding (i.e. assigning variables at run time) and polymorphism (objects reacting differently to identical messages) should put the reader in good stead for reading this text.

The effectiveness of using object modules for system development, will be focused around the IMPW throughout this text. The size of this project is substantial (approximately 100k lines of code) for allowing us to view the benefits and faults of the Object Oriented approach in medium to large software projects.

Chapter 2

Objective-C

2.1 Introduction

The commercial marketplace for Object Oriented Languages has increased steadily over the last five years. Magazine adverts now offer a number of Object Oriented environments for software development. The benefits of reusable, more reliable and easily extendable code is offered by a number of Object Oriented suppliers. Smalltalk, Eiffel, XLISP, C++, Objective-C and Objective-Pascal are examples of some of the many programming environments which can be used for Object Oriented development.

Before any tools were designed for the IMPW, a decision was taken to establish a software environment for tool development. The final decision made by the management team was to develop using the Objective-C language. The reasons for this choice are numerous, i.e. reputation of the software environment, facilities offered by the language such as libraries containing object modules for graphics and input/output capabilities, portability of software on graphic workstations, plus all the various political and financial reasons which are associated with a project. No justifications for the choice of language is given here, instead the objective (excuse the pun) is to explain the features of the Objective-C language and the syntax required to use these features.

2.2 Objective-C data types

The Objective-C language is a superset of the C language. Using Objective-C does not necessarily mean learning a new language. C users only have to adopt a new approach in the use of a familiar language[Cox 86]. The language adds precisely one new data type the object identifier referred to as type `id`, to those provided by standard C. This declaration is made in a manner similar to other C types.

```
int    i;           // declared an integer
char   *c;          // declared a pointer to a character
id     myObject;     // declared an object identifier
```

Figure 2.1: Object Declaration

The variable `myObject` may be used to identify any type of object. The amount of space required to identify object labels, similar to `myObject` is a constant, but the space occupied by the object will depend upon each objects private data (instances variables). There are two types of object which the Objective-C user may use, class or instance.

2.2.1 Class and Instance Objects

In chapter one it was stated that the shared part of the object is called the class and the private portion containing the unique data is called the instance. The class which provides the mechanism for sharing methods is regarded as an object by many languages[Tho 89], including Objective-C.

The class may be thought of and is sometimes referred to as a **factory object** producing new objects similar to itself. Each new object created by the factory class is an instance of that class, the data store in each object is private

to that instance. Further reference in this text to an instance can be regarded as an individual object, which has been created by some class. The term class is also a reference to an object, but this object is used to define the shared parts of similar objects or factory objects.

2.3 Messages

When an object wishes to perform one of its operations, it sends a message to itself. People accustomed to conventional programming techniques find this terminology difficult to grasp at first, what exactly do we mean when we talk about sending messages?

message : *Defⁿ* message sending is used by object oriented languages to make an object perform one of its operations.

Messages may be thought of as something similar to function calls. They can contain a variable number of arguments and execution of program code is halted until the called procedure finishes. It is possible to return any data type similar to that returned by functions in C. The examples below illustrate the Objective-C syntax for sending messages.

message	selector	argument
Point new	new	none
anObject new:5	new:	5
anObject give:me:5	give::	me, 5

Figure 2.2: Message Syntax

Unlike conventional function names, message selector names do not guarantee unique names throughout the system. This is because the selector name is

also dependent on the object that received the message.

In order to determine which procedure(method) belongs to which object, each object is allocated space to store a table which contains pointers to all the procedures which are private to its domain. This table is known as the dispatch table and the pointers in each slot are known as the message selectors.

The Objective-C syntax for sending messages is summarised below.

```
result = [anObject doSomething: argument];  
          (receiver)   (selector)  
          (object)    (message)
```

Figure 2.3: Message Glossary

The text string after the left brace in any message expression will always represent the object which is receiving the message. Objective-C uses the convention of presenting the first letter of a class/factory object in capitals. Instance objects begin with lowercase letters therefore `anObject` referenced above would reference an instance object. Selecting message names is similar to selecting C function names. Names chosen should relate to the task at hand; a convention which should be performed by all programming languages. However instead of using underscore to concatenate words in a function name, Objective-C programmers use the convention of representing the first letter of each word, after the first word, in capitals as below.

```
ac = add_interest_to_account();    //C  
ac = anObject [addInterestToAccount]; //Objective-C
```

Figure 2.4: Objective-C Message Conventions

Colons are used to separate arguments and also form part of the selector's name (see figure 2.2). The message statement is always terminated by a right brace. The Objective-C syntax also allows programmers to embed message calls inside other messages and ordinary C function calls. Statements as in figure 2.5 appear frequently in many Objective-C programmers code. In the first example the factory method `Book`, creates a new book and then adds the object to `bookCltn`. Note the order in which the methods are performed follows C precedence rules. Messages are executed from left to right, with inner nested messages taking precedence. In the second example the fifth book is taken from the book collection (all collections begin at zero), the book then finds out what size it is before printing it's result. The third example uses the C `sprintf` function, the string and integer values are provided by `title` and `pagesize` methods respectively.

```
[bookCltn add:[Book new]];  
[[[bookCltn at:4] size] print];  
sprintf(fred,"%s %d",[aBook title],[aBook pagesize]);
```

Figure 2.5: Objective-C Statements

2.4 Methods

The message or selector names for all the procedures which `anObject`(see figure 2.3) can perform are kept in a dispatch table in `anObject`'s shared part(class). The dispatch table uses a selection mechanism which indicates the appropriate operations `anObject` should respond to. In Object Oriented languages the operations carried out by objects are called methods. These methods are similar to ordinary functions, i.e. they can have any number of various argument types, they can return various types of results and the operations in both is determined by the code inside the method or function. However there are a few differences between functions and methods:

- Method names are not unique, i.e. different classes may have the same method name.
- A method needs to address additional data space i.e. the private data inside the object sending the message.
- Methods are called indirectly by messaging.

The Objective-C language permits two basic method types which can be activated by sending messages to the appropriate factory or instance methods. Factory methods are the operations performed by the class object such as initialisation of instances. Instance methods relate more to specific operations which an instance of a class should perform, such as printing its contents. The space, data types and operations performed by these objects are determined in the class definition file.

2.5 Class Definition Files

Objective-C classes which provide the basic unit of modularity in an Object Oriented system are created in the class definition file. This file determines the class name, its hierarchical position relative to other classes in the hierarchical network, the objects instance variables and the methods which may be performed by the class or instance objects.

The class definition listing in figure 2.6 represents the class Room. As well as the six instance variables declared for Room, the variables from the inherited classes back to the root class are also added. The root object in the Objective-C inheritance tree is the class Object, which is the only Objective-C class that has no parent or super class. The inheritance path for all objects always ends with this abstract object. The immediate class from which Room inherits is the Object class, therefore only instance variables and methods may be inherited from this class. The class Object adds one instance variable to the Room class, this variable is known as the *isa* pointer. The *isa* pointer is important, as it is used by objects to form an inheritance chain to the root object.

The methods which surround the Room object's private data complete the

rest of the listing. Methods with plus signs("+") before the selector name represent factory(class) methods. These methods are only accessible to an object's factory methods, a common fault when beginning Objective-C programming is asking an instance of some class to perform one of its factory methods. Unless there is an instance method with the same name in the object class or one of the inherited classes an error will occur. The minus sign("-") before the selector name represents instance methods. Instance methods may only be accessed by sending messages to instances of that class or sub-classes.

The values returned from `floorspace` and `colour` are of type `int` and `char` respectively. When the method fails to give a return type, the default value `id` (object identifier) will be given, unlike C whose default type is an integer. The `"create"`, `"length:"` and `"breadth:"` methods are examples of methods which return type `id`. Note that colons used to separate method arguments, must appear in the selector name. If the Room class calls a method which cannot be found in its class definition file the search is repeated in the parent class. This procedure is repeated until the called method is found or until the inheritance chain is exhausted in the root Object.

```
/* Objective-C source file for the class Room */
= Room : Object (Demo, Primitive)
{ // instance variables of class Room
    int length;
    int breadth;
    int height;
    int windows;
    int doors;
    char *colour;
}
// Create a new Room by using the superclass 'new' method
+ create {
    id newRoom; // declaration of local variable
    newRoom = [self new]; // creates new instance of room
    [newRoom length:40];
    [newRoom breadth:60];
    [newRoom height:5];
    [newRoom windows:1];
    [newRoom doors:1];
    [newRoom colour:"blue"];
    return newRoom;
}
// Sets the room length variable for a Room object
- length : (int) roomlen {
    length = roomlen;
    return self;
}
// Sets the room breadth variable for a Room object
```



```

- breadth : (int) roombreadth {
    breadth = roombreadth;
    return self;
}
// Sets the room height variable for a Room object
- height : (int) roomheight {
    height = roomheight;
    return self;
}
// Sets the room windows variable for a Room object
- windows : (int) roomwindows {
    windows = roomwindows;
    return self;
}
// Sets the room windows variable for a Room object
- windows : (int) roomwindows {
    windows = roomwindows;
    return self;
}
// Sets the room doors variable for a Room object
- doors : (int) roomdoors {
    doors = roomdoors;
    return self;
}
// Sets the room colour variable for a Room object
- colour : (char *) roomcolour {
    colour = roomcolour;
    return self;
}
// Determines the integer size for floor space
- (int) floorspace {
    int floorsize;
    floorsize = length * breadth; // length and breadth refer
    return floorsize;             // to objects private data
}
=:                               // symbol which terminates the class

```

Figure 2.6: Room Class Definition File

Classes such as Room (or any other Objective-C class) are executed directly by a main program or indirectly by another class. The main program for initiating Objective-C class is similar to its C counterpart, with some additional features. The Objective-C main program must always declare the message groups used by the file.

Classes and messages are declared at the bottom of the main module so that the compiler can combine all the classes and messages used by the program.

The syntax for classes is simply `@classes(class list)` where the list contains all the classes referenced in the file. The syntax for message is similar `@message(message list)`, however so long as the messages are in the same file as the classes statement, the message groups do not have to be listed.

2.6 Self and Super

To prevent losing track of objects while sending messages through the inheritance chain, the keyword **self** is used in Objective-C, to identify the object receiving the message. The word **self** in the Room factory method "create", is the receiver of the factory message "new". Methods which return the value **self**, simply return the object which received the message. Whether the objects private data values have changed since the message was sent will be dependent solely on the called method's code. The methods in the Room class listing which return **self**, initialise the instance object `newRoom` with values `newRoom length 40`, `newRoom breadth 60`, `newRoom Height 5`, `newRoom window 1`, `newRoom doors 1`, colour blue.

Besides inheriting instance variables, Objective-C classes also inherit factory and instance methods. When the inheritance tree has to be followed back through a number of parent classes to find a method, the pseudo-variable **self** identifies the object which received the message. The Objective-C variable **super** is similar to the **self** variable. However unlike **self** where the search for a selector name begins in the object's dispatch table which received the message, the search of selector names begins in the object's parent class when the receiver is **super**.

```
+ new {  
    id myRoom;  
    myRoom = [super new];  
    ...  
    return self;  
}
```

Figure 2.7: Method Structure

If the factory method "create" in room was replaced with the factory method "new" as in figure 2.7. The receiver of the "new" message is **super**, so instead of searching for the "new" method in the class dispatch table where the message is sent, the search commences in the parent class (which in this example would be Object). Using **self** instead of **super** in this method would put the program into an infinite loop, an error often encountered by naive Objective-C programmers.

2.7 Objective-C Inheritance Library

The inheritance mechanism provided by some Object Oriented languages (Ada and Modula-2 are examples of Object Oriented languages which do not provide inheritance) encourages reusing existing code. The inheritance mechanism may not be a required feature of object oriented languages, but it is certainly a very desirable one. Inheritance helps in two important system development principles.

Reusability — the ability to produce software that may be used in many different applications.

Extendibility — the ability to add code to existing source, without modifying the old source.

The hierarchical inheritance mechanism used by the Objective-C language contains a number of generic classes linked to a root object. Each subclass becomes more specific as we follow its path from its parent class. All the instance variables and methods from the parent class are inherited. This means that a subclass does not have to redefine methods and instance variables from its parent class. A subclass can be viewed as a more specialised version of its parent class i.e. it is similar, but with a few extra bits added on (extra methods and instance variables). When similar method names exist in various classes, the method in the lowest (more specific) subclass will override the parent method. Access to methods further up the inheritance tree are referenced using the variable **super** mentioned above.

2.7.1 Foundation Classes

One of the strong selling points of the Objective-C language, is the number of software classes that are supplied with the compiler. These classes provide the foundation from which software development can begin. The classes provided, help form the class inheritance foundation library as shown in figure 2.8. The order of class inheritance is indicated by indentation below the respective super-class.

The foundation classes provided are divided into three message groups, which must be declared with the class, so that the compiler can store all the method types returned. The classes provided in the Objective-C foundation library belong to one of the following three groups mentioned below. Classes created by the user are added to their own message group, in the Room example, the class is attached to the message group Demo.

- **Primitive** classes which provide a basic foundation for building software-IC's. The term software-ICs[Coxb85] is a term used by the Objective-C suppliers to reference reusable classes, this is analogous to the way hardware builds from existing integrated circuits(IC).

- **Collection** classes which provide functionality for maintaining different types of collections.

- **Geometry** classes which provide the basic components for building graphical user interfaces.

To explain the operations, data and foundation grouping of each of the above classes in detail, would require time and space outside the bounds of this thesis. Readers interested in finding out more about Objective-C and the classes provided with this environment should refer to the Objective-C manual[OC - 87]. The section below briefly describes the role of some of these predefined classes to give the reader a taste of the power provided by this environment. However it must be stressed, that in order to build system applications using this environment, it is important that the user has a good understanding of all available classes.

OBJECT \Rightarrow positioned at the root of the inheritance hierarchy has its capabilities inherited by all other classes. The root Object can be thought of as an abstract class, which means that its contribution is not really directed towards creating instances of this class, but more towards providing services which can be used by objects further down the inheritance path.

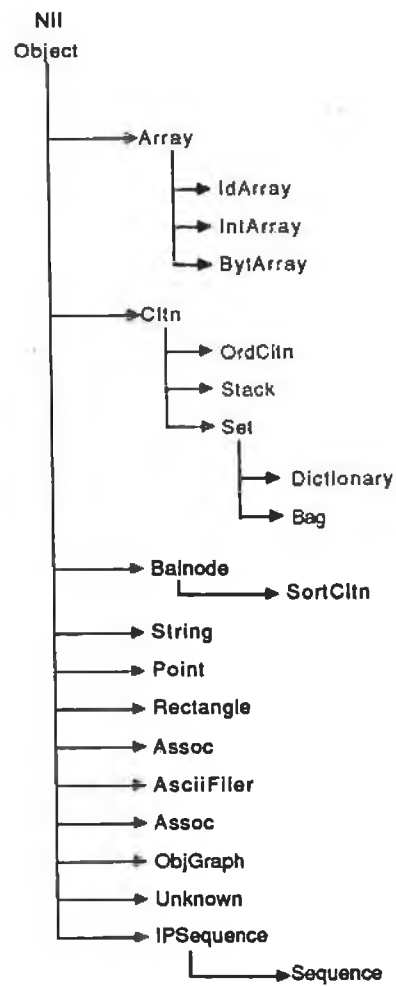


Figure 2.8: Objective-C Hierarchical Inheritance Structure

The class `Object` defines one instance variable, the `isa` variable which is a pointer to an objects shared part and is automatically inherited by all sub classes. This allows all instances of classes to point at the factory object, from which they will begin to inherit.

The methods provided by the `Object` class facilitate features such as storage/retrieval of objects in memory, error handling, comparisons, object interrogation plus other features whose inclusion, would be general enough to be used by any class.

ARRAY \Rightarrow This class is the abstract superclass of all array classes. It is designed to support random access to indexed instances variables. The more specific sub-classes are used to hold different array types.

IdArray	\Rightarrow	stores object identifiers
ByteArray	\Rightarrow	stores chars
IntArray	\Rightarrow	stores integers

Methods are provided for resizing, accessing, comparing, sorting etc. However these classes do not provide complete flexibility, because array size must be defined before they are created. Changing the size of an array object is possible, but by no means straight forward(i.e. the array and its contents must be tracked down and relocated in memory).

ASCIIFILER \Rightarrow This class provides a general technique for having persistent objects. Allowing objects created by the system to outlive the execution time of the system. The crux of these operations is to convert binary values into textual representations and vice versa. This class saved a large amount of time and code during IMPW development, removing the need to write code to store and retrieve objects in memory.

2.7.2 Collection Classes

Development of the IMPW discussed extensively throughout this text relied heavily on the facilities provided by the collection classes. Unlike the Array classes, collections provided the flexibility of variable length collections. The collections are used for holding objects, some of which may be types of collection themselves.

CLTN \Rightarrow is the most abstract of the collection classes, whose role is mainly to provide the functionality required for more specific collection classes. This class includes methods for adding, removing, testing and memory allocation. This abstract class is rarely used for the creation of collections, instead that responsibility performed by one of the sub classes.

ORDCLTN \Rightarrow subclass of Cltn, keeps the objects entered to the collection in order, no nil entries are permitted in the collection. Statements required for setting up an order collection are given below.

```
anObject = [OrdCltn new];
```

Note that the size of the collection does not have to be specified, as it will be adjusted automatically as objects are added.

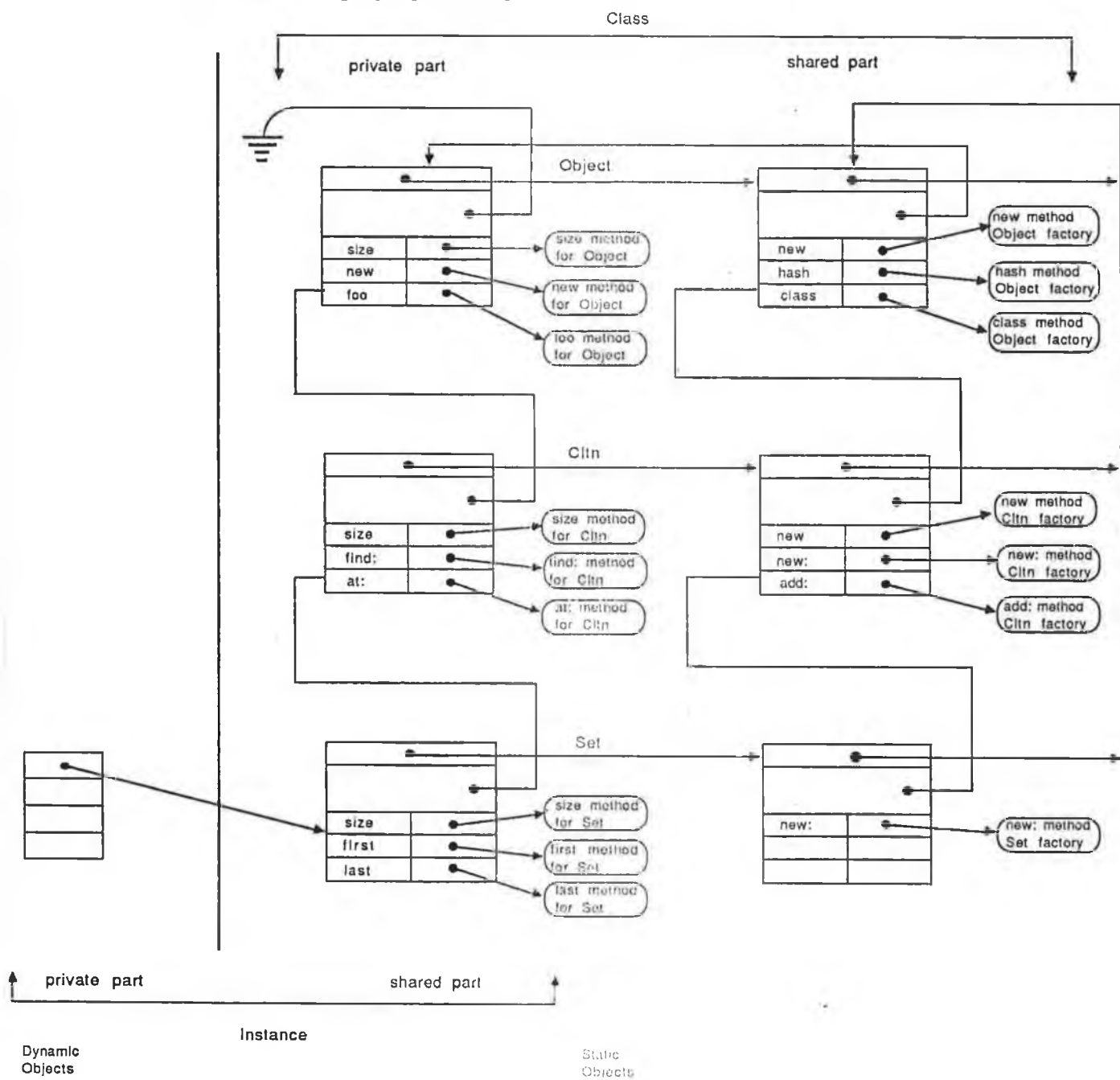
SET \Rightarrow are collections which disallow duplicate entries, a common use of such objects is in the creation of symbol tables. Sets place all objects added to them into a hash table. The assumption is made that after objects are added to a set they will not change anything about themselves.

2.8 Tying it all together

The diagram in figure 2.9 shows how the objects(classes and instances) are arranged in the Objective-C environment. Due to the number of methods inherited by the class objects, it is not possible to represent all methods. The slots below each of the boxes represent the dispatch table associated with an object. The objects in the left hand column represent the shared portion used by the instances. The right hand column represents the shared portion used by the class objects.

What surprises many people is the fact that the private part of one object is the shared part of another. By simulating sending messages in this architecture, any mystique surrounding it may be removed. For example if the message [self foo] is sent by an instance of Set, the dispatch table for Set instances is searched. If the selector is not found, the search is repeated in its parent class Collection and then in the Object dispatch table where it is found.

The meta class data structures(the classes shared part) on the right contains the dispatch tables that are searched if a factory message is sent. Notice how



```
aSet = [Set new];
[aSet foo];
```

Figure 2.9: Objective-C in Memory

the name and function of the block changes when the point of view is changed by selecting a different object. If a message is sent from a factory object, the previous shared part now becomes the private portion.

2.9 Objective-C Compiler

While not an important issue for Object Oriented implementation and design, details of the Objective-C compiler mechanisms have been added for completeness. The compiler can be thought of as a program which translates Objective-C source code, into a binary executable form, for execution on your machine. The steps required to achieve the executable code involves steps similar to ordinary C compilations. However unlike conventional compilers dealing with files in isolation, compilers with inheritance capabilities cannot accomplish this. Changes to classes during system development, causes information to flow in both directions between the compiler and the library. This leads to some subtle differences in the way compilations and linking is managed.

The Objective-C compiler contains a control program, which coordinates the various tools and the files manipulated by these tools. The Objective-C control program, executes tools in a tool chain when compiling the Objective-C language. The tool chain must include

- C Preprocessor
- Objective-C Translator
- Target C compiler
- Target Linker

to perform Objective-C compilations.

The first phase of the Objective-C compilation process involves the preprocessor. The source files, containing class definitions, are the input for the preprocessor. The preprocessor performs a translation pass, replacing `#include` and `#define` statements with original source and definitions for respective macros. The output from the preprocessor, the original source with include files merged

and definitions expanded, is used as input for the second phase.

At the second phase, the Objective-C translator converts the expanded Objective-C source code into C source code. The Objective-C compiler translates class definitions, class references and message references into C source. Ordinary C statements travel through this phase untouched.

During the next phase the target C compiler translates the source code into relocatable machine code. The relocatable object file is produced by the target C compiler. The standard librarian may place this file in memory or the linker may be used if an executable image is required.

The final phase in the tool chain uses the linker to combine the object modules as specified by the source. Objective-C class libraries, C library code plus the user's own class library and library functions may be linked into an executable program. The output from this phase will be the executable program.

The suppliers of Objective-C supply the C-preprocessor and the Objective-C translator. The host environment must supply the target C compiler and the target linker.

The command syntax required to compile Objective-C files is

```
objcc [options] filename(s)
```

The options are similar to those provided by the C compiler

- c produces relocatable object modules
- o produces final output file, a.out is used if no option specified
- g debugging information

To ensure consistency, a common set of routines manage all diagnostics issued by the compiler. All the Objective-C errors have (OC) printed before them. If the diagnostic does not contain an (OC) then the problem relates to conventional C code.

2.10 Summary

This chapter has given the user an express trip around the Objective-C environment. Explaining all areas of the language was not possible within the context of this thesis. The examples and discussions given are used mainly to give the reader a broad understanding of the Objective-C language and help ease the readers passage through this text. As with most languages, the best way of learning Objective-C issues discussed in this chapter and more, is through practical experience.

Chapter 3

Risk Analysis Tool

3.1 Introduction

The IMPW discussed in the first chapter, divided the workbench into three main areas or objects of functionality. The Information System(IS) is the workbench nucleus, verifying and storing data, which affects the wide range of variables associated with project management. Interactions with the IMPW are controlled by the Manager Workbench Interface(MWI). This object is responsible for presenting data and information, generated by the tool, to the end user in a format that is easily understood and manipulated. Between these two high level IMPW objects the Computational System(CS) resides. This object contains the tools which allow the IMPW to be used by the end user. These tools add and manipulate data in the IS database, determining the values presented by the MWI. The goal of the overall IMPW project was to produce a prototype workbench, containing a comprehensive and integrated set of tools, to help with the management of medium to large-scale software development projects.

Discussion during this and the next chapter, will be centered around the functionality provided by two of these tools, mentioning briefly how the Objective-C environment was used in their development when appropriate. Due to the complexity and size of these tools, it is impossible to describe them in any depth inside this thesis. The code illustrations given throughout this text show only a small proportion of the statements, and data variables common to these tool environments. However code listings associated for the creation of both Risk and Calendar tools may be viewed in Appendix D and E respectively. This chapter concentrates on the Risk Analysis tool, while the next is dedicated to the

Calendar tool.

3.2 Risk Tool Overview

The Risk Analysis tool was designed to help the project manager “walk around” the project at an early stage, and anticipate potential major sources of risk to the project. The tool attempts to quantify risk under a number of headings and to propose practical steps that the project manager could take to reduce or offset identified risks.

3.2.1 Tool Input

As a result of a literature review and consultation with a number of experienced software project managers, four “risk management areas” which the risk tool should address were identified [Boe 81],[Zal 77], [Dav 82],[Alt 78],[Cha 85],[AFSC87]. An initial set of twenty three project risk areas (riskdrivers) were identified, which could be expected to contribute to the level of risk in the project in one or more of the risk management areas. The project managers judgement of the values of the riskdrivers for the project constitute the primary input to the Risk Tool. Great care was taken in the design of the rating scales to be used to ‘quantify’ the levels of the riskdrivers. The four risk management areas are defined in Appendix A. The riskdrivers and their values are defined in Appendix B.

3.2.2 Tool Output

The major output from the tool is the “risk report”. This report is in two sections. The first section provides the “risk measures”, one for each of the four Risk Management Areas. The risk measures are computed from the riskdriver values provided by the project manager. The algorithm used to compute the Risk Measures is described in the “Measure Class” section. The second section of the risk report consists of a set of text “advice paragraphs” suggesting risk reduction strategies to the project manager. The computation of the “advice paragraphs” is based upon the application of production rules (i.e.the “risk analysis rules”) to the values of the riskdrivers. The prototype tool contains only thirty Rules, thus the level of “wisdom” it displays is limited. Some of the Rules were formulated through discussion with project managers. Other Rules were gleaned from the

general project management literature. The manager is able to interrogate the risk report to establish the basis on which the risk measures, and the particular advice paragraphs shown were reached. The procedure the tool uses for this “explanation” function is outlined in the “Rule interrogation” section.

3.3 Important Classes

Before explaining the functionality provided by the Risk class, it is important first to introduce the main classes and the private data(instance variables) associated with these structures. The Risk tool initiated the development of a number of new classes. Four of the most relevant classes for understanding terminology relating to functionality to be described later in this chapter, are described in the following sections. The classes to be discussed are given below.

- Risk
- Rule
- Text
- Measure

3.3.1 The Risk Class

The various risk areas used by the tool are referred to as Riskdrivers by the Risk Analysis tool. Reference throughout this text to Riskdrivers refer to areas of software risk. The Objective-C mechanism for representing a Riskdriver object is shown in figure 3.1. This declaration is taken from the Risk class definition file Appendix D. Because no standard information on the construction of riskdriver objects exist, the object structure for risk objects was made from the accumulation of information, referenced in the “Tool Input” section and other personnel involved in the development of the IMPW.

The data values which are common to all riskdrivers are mirrored by the instance variables in the Objective-C factory class Risk. The instance variable riskdriver indicates the name of the risk area eg. “Scale of Project”. The risktxt

```
// Risk Factory Class
= Risk : Object (RiskGroup, Collection, Primitive)
{   // instance variables of risk
    char * riskdriver;
    char * risktxt;
    char * riskcondition[6];
    int    riskweight[6];
    char * riskhelp;
    char * attrName;
    char * entityName;
}
```

Figure 3.1: Instance Variables for the Risk Class

instance supplies supplementary information, which is used to make the reading of riskconditions more comprehensible, when displayed on screen. The Risk class allows each Riskdriver to have a maximum of six riskconditions to represent the different characteristics it might have.

The data types used here were stored in an array, a more flexible structure would have been obtained, if riskconditions and riskweights were stored in an object of type collection. The collection class would have eliminated the need to restrict riskcondition and riskweight to a specific size, as the collection object could have expanded automatically for variable riskcondition numbers. All riskconditions associated with a Riskdriver are given a risk weight, which is stored in the corresponding riskweight array. The weight values are used for estimating the percentage project risk for all riskdrivers used by the Risk Analysis tool. In order to clarify the risk areas referenced by some Riskdrivers, additional information is supplied in riskhelp. The instance variables attrName and entityName are present in the Risk class to identify the IS table and the column name where the Riskdriver values are recorded in the IS database.

3.3.2 The Rule Class

The output from the Risk Analysis tool has been specified as a report detailing numeric measures of risk, plus textual information explaining what is causing the

risk and advisory text paragraphs indicating corrective action. The presentation of this textual information relates to "risk analysis rules", entered by an expert on risk analysis for software projects. The rules represent riskdrivers having riskcondition values, which may affect overall project success. The syntax of these rule messages may indicate risk associated with one or more risk objects(riskdrivers).

Creating a factory class to represent rules indicating project risk based on the values of the riskdrivers, introduces problems similar to those encountered when setting up the Risk class. When setting up the Rule class, we need to know what data values should be used to represent the Rule, what methods to associate with Rule objects and what class should the Rule class inherit from. There is also the additional problems of linking rule objects to risk and text objects.

Various object designs were considered for the internal representation of the Rule class such as linked lists, B-trees, binary trees, arrays etc. The approach chosen to model the Rules is shown in figure 3.2.

```
= Rule : Object (Riskgroup, Collection, Primitive)
{
    char * rules[6];
    char * condition;
}
```

Figure 3.2: Instance Variables for the Rule Class

In the prototype version, six is taken to represent the maximum number of riskdrivers and condition numbers allowed to relate to any condition number result. The condition number which represents the link between Rules and associated text is represented by the character pointer condition.

The table in figure 3.3 shows how instances of Rule are stored in the Risk Analysis tool. Each row in the table represents an instance of the Rule class. The entire table represents a collection of rule instances which is called the ruleCltn in the Risk tool. The asterisks denote the number of riskdrivers referenced by a Rule when less than the maximum six in the prototype. Again, as in the Risk class, "collections" rather than "arrays" would have provided a more flexible way of holding Rule operands as no upper bounds were required. However to

Rules						Condition Number
R1.1	R2.1	*				C1
R3.2	R9.1	*				C2
R5.4	*					C3
R4.2	C1	*				C4
R7.2	*					C5
R7.1	R8.1	*				C5
C2	C3	*				C6

Figure 3.3: Storing Rule Objects

keep the Rule syntax relatively simple, arrays with a maximum limit of six were used to store Rule operands in this prototype version. The values in each cell reference riskdrivers, the value R1.1 in the top left cell references riskcondition one for riskdriver number one, the value R3.2 represents riskdriver number three riskcondition two etc. Some Rule instances also contain values with a C preceeded by an integer value. This is used to represent rules from previously defined riskdrivers, as well as providing a link between rule and text objects.

e.g. C1 = IF R1.1 AND R2.1

The value C<integer value> at the end of each row is called the condition number. This number will link risky riskdrivers to their associated diagnostic text. The condition numbers, which are used to relate associated text are generated automatically by the tool in an ordered sequence, designed to simplify searching, when performing operations on these objects. It also helps to avoid the situation of the SuperUser relating a rule to an undefined condition number, thus reducing the level of validation required on rule objects. The interaction between riskdriver values and Rule sentences is accomplished using a rule specific grammar syntax as shown in figure 3.4.

Rule Validation Using this rule syntax as an intermediary between riskconditions and high level rule sentences introduces other problems. Some type of validation must be made to ensure that rules entered conform to a syntax which

```
IF R1.1 AND R2.1 THEN C1
IF R3.2 AND R9.1 THEN C2
IF R5.4 THEN C3
IF R4.2 AND C1 THEN C4
IF R7.2 OR R7.1 AND R8.1 THEN C5
IF C2 AND C3 THEN C6
```

Figure 3.4: Sample Rules

relates rules to riskdriver objects used by the Risk Analysis tool. Checks must also be made to ensure that riskdrivers referenced exist, riskconditions referenced to specific riskdrivers exist and that condition numbers entered in rules already exist (i.e. the results of previously defined rules). These problems highlighted the need for a number of validation procedures to be attached to the Rule class.

The validation of the rule syntax can also have an effect on the efficient running of the tool. If objects are to be created each time the "risk expert" is about to enter Rules, what happens when the Rule entered is invalid? It is possible using the Objective-C mechanism to create objects which know how to destroy themselves. However the creation and then deletion of an object would seem wasteful of machine and user time. After analysing these factors, the approach for creating Rule instances was finalised.

Instead of creating rule objects before the rule sentence was entered by the "risk expert", the rule sentence is validated first, before the creation of Rule instances. Functions were required to parse the sentences into operand and operator stacks and check the validity of the grammatical sequence. Checking of operands ensures that riskdriver and riskcondition numbers referenced exist and that condition numbers forming part of a Rule already exist as a result of previously defined Rules.

Statements containing the OR operator signifies that more than one instance is related to a Rule sentence. The Rule class only creates Rule instances, if the entire rule syntax entered is correct. This decision relates to the automatic sequencing of numbers. Entering the invalid part of a rule at some later stage would require the manual entry of the condition number, plus extra functions to ensure that it is valid. These checks however could not correct the possibility of the SuperUser entering the wrong condition number for the rule.

By using ordinary C functions instead of Objective-C methods, verification of Rule syntax may be performed before Rule objects are created. This highlights one of the advantages of the hybrid languages such as Objective-C over a pure Object Oriented language such as Smalltalk, where all operations must be performed by the object(i.e. the object must be created before it can be validated).

3.3.3 The Text Class

The second part of the risk report produced by the Risk Analysis tool, prints textual information related to the current project characteristics. This textual information is divided into two sections. The first part containing an explanation why the project is deemed risky with its current values. The second contains advice text indicating corrective action.

The class Text is used to store and print this textual information. The text associated with an object is related to Rules set up by the SuperUser and triggered when project characteristics relating to riskdrivers match these rules. The association between the Rules and the text is formed by the following code.

```
condition_num = atoi([aRule condition])
someText = [txtCltn at:condition_num]
```

The condition number for the object aRule is converted from character to integer, before assigning it's value to the integer condition_num. This variable is then used to access the text object relating to this rule. All text objects are stored sequentially in an ordered collection(txtCltn). Using the "at:" method the related text for a set of Rules may be stored/retrieved easily. The diagram in figure 3.5 shows pictorially how these objects are related.

Despite its strong links with the Rule class there is no inheritance between these two classes. Making Rule the parent class of Text was considered while developing the Text class. The textual information could have been related to Rules, keeping the bond between these classes very tight. However I was afraid of creating large cumbersome objects, containing redundant variables and methods which were not really required. I also envisaged problems with operations such as print. If the print command was sent by a Rule object the textual information could not be obtained because it was at a lower inheritance level. The Objective-C inheritance mechanism works upwards from the object, sending the message towards the root, not downwards.

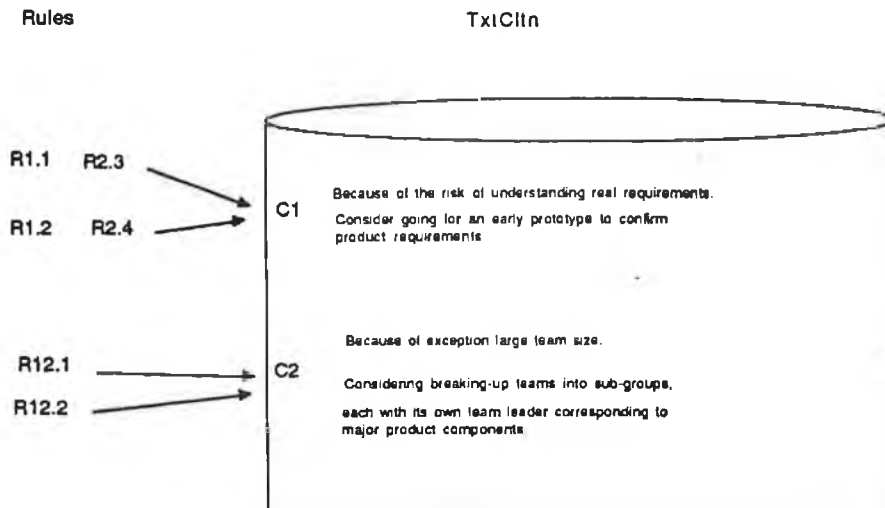


Figure 3.5: Rule and Txt objects.

3.3.4 The Measure Class

Printing risk metrics for software projects, indicated the need for a class which would perform some type of computation on riskdriver characteristics. The factory class Measure is used in the tool for calculating software risk and displaying the output as a percentage value. The class calculates project risk for the four risk management areas as a percentage. High percentages indicate a high probability of project failure in that area.

- [1] Cost/Schedule Failure
- [2] Premature Project Termination
- [3] Product Functional Failure
- [4] Product Technical Failure

Each riskdriver when created must be associated with one or more of these risk areas. The Risk class method "create" will elicit the required management areas for each riskdriver before adding it to the risk collection. The management areas associated with each riskdriver are shown in Appendix C.

The calculation for estimating risk in each of the risk management areas is given below.

$$\text{Risk measure} = \frac{S_i}{\text{Max}[P_i]}$$

where

S_i = Riskdriver weighted value associated with riskcondition selected

Max[P_i] = Highest possible weighted value for this riskdriver

The current algorithm for calculating risk gives a crude estimation of risk. More sophisticated algorithms may be integrated into future tool versions. The only code that needs to change will be the method containing the risk calculation algorithm, thanks to the encapsulation features associated with Objective-C classes.

Before printing risk percentage for the four risk management areas, the calculation method is always performed. This process is necessary so that changes to riskcondition values, are reflected in the percentage output of risk management areas displayed to the tool user.

3.4 Types of RiskTool Users

The functionality provided by the Risk Analysis tool is designed to meet the needs of two types of user. The software project manager(sometimes referred to as the user), for whom the tool is specifically built and an expert on software projects responsible for entering risk related attributes. The input required from the project manager will be the relevant project characteristics which closely match specific riskdrivers. The output expected will be as described by the tool output above.

If this prototype version is to evolve, it is important to have a system which will facilitate these changes. The addition of new riskdrivers and Rules must be considered a feasible possibility if the tool is to mature. The addition of such

details for use by the project manager must be performed by an expert in risk analysis in software projects. Throughout the rest of this text, this person will be referenced as the “SuperUser”.

With conventional languages the addition of new data affect the original code, eg. array sizes are reset or case statements adjusted to accept a new type. Because of the facilities offered by Object Oriented languages such as encapsulation, dynamic binding and inheritance, it is possible for the SuperUser to add additional riskdrivers and Rules without causing any recompilation of existing tool classes.

3.4.1 SuperUser Functionality

Before providing the operations required by the project manager for analysing risk, the functionality for entering riskdrivers and Rules had to be developed. We have seen the structure the Risk instances which can be developed in the Risk factory class. Now it is time to explain the operations which are required to enter and maintain these objects in the Risk tool. The main areas of functionality required by the SuperUser are listed below.

- Add
- Delete
- Amend
- View
- List

• **Add** When adding riskdrivers to the Risk Analysis tool an instance of risk is created by the Risk factory class. The new instance of Risk will inherit all the necessary variable data types. The initialisation of these data variables must be entered from a keyboard or via a file. The more direct approach of entering data variables at the keyboard was adopted for the Risk Analysis tool, to allow the SuperUser to spot erroneous input. The SuperUser enters the relevant data in response to screen prompts, sent by the Risk create method. An example of the Objective-C code for adding riskdrivers and the type of riskdrivers created are given in figure 3.6 and figure 3.7 respectively.

```

+ create {          // Create a new riskdriver for
                    // Risk Analysis Tool
    id risknum;
    int weight;
    int i = 0;
    self = [self new]; // an instance of Class Risk is returned
    system("clear");   // Unix call to clear screen
    printf("Enter Riskdriver Title ");
    getrisk();
    riskdriver = malloc(strlen(iobuf));
    strcpy(riskdriver,iobuf);

    .....
    .....
    return self;
}

```

Figure 3.6: Initialisation of Riskdriver Instance Variables

SCALE OF PROJECT (NO. OF PEOPLE)	→ riskdriver
In relation to what we are accustomed, the size of the project is -	→ risktext
at least three times as big	4
about twice as big	3 → riskconditions
about the same size	2 → riskweights
smaller	1
Nil Help	→ riskhelp

Figure 3.7: Riskdriver Instance

If for some unforeseen reason, changes are made to the instance variables in the Risk class, these changes would also have to be made in related methods causing maintenance and recompilation. The importance of this discussion highlights the need for careful analysis when developing instance variables for class objects. Despite all the advertising hype about "Object Oriented language only extending not amending code", nothing can be done if the programmer defines invalid instance variables for a class object. Classes can be written for converting objects when the classes instance variables have been changed, however methods affected by the addition or deletion of instance variables must also be updated. The instance variables shown in earlier figures are those used by the prototype workbench and accepted as correct by the IMPW steering committee. With this assurance, it was possible to concentrate on creating classes and methods for the Risk tool, which would be as robust and reusable as possible.

The Objective-C factory method "create", performs the necessary initialisation for each Riskdriver by extracting risk characteristics at the keyboard. When the initialisation process commences, a new Riskdriver is created by the [self new] message. This method is inserted at the beginning of the "create" method to setup the risk template object required. The receiver of the new message "self", identifies the factory object Risk. Typically this factory method creates a new instance of its class. For efficiency reasons programmers may change the identity of self to an instance eg. self = [self new]. When this happens, all the messages subsequently sent to self are sent to the newly created instance not the factory.

On entering all the risk characteristics, the create method prompts the SuperUser to enter the risk management areas. Each riskdriver created should be associated with at least one of these areas see Appendix C. After eliciting all the necessary data to represent a riskdriver, the new instance is stored in a collection of riskdriver instances (called the riskCltn in the Risk Analysis tool).

• **Delete** This functionality is provided to allow the SuperUser to remove riskdrivers from the Risk Analysis tool. Unlike adding riskdrivers into the Risk Analysis tool, where adding new instances is simple once riskdriver characteristics are known, deletion of riskdrivers is however a more complex operation. The removal of riskdriver instances from the Risk Analysis tool may involve the removal of instances from the Rule and Text classes, if instances of these classes are associated with the riskdriver marked for deletion.

The deletion of riskdriver also leaves gaps in the collection object holding the instances. Storing riskdrivers in a collection with gaps is not only wasteful of space, but could also affect the way searching and printing of objects is performed.

Compressing the collection after deletion would seem a more sensible option, but this means updating riskdriver numbers. Therefore the Rule values which apply to these riskdrivers and the resulting condition text object must also be updated. This solution is however more acceptable than having a collection containing gaps. Problems which spring immediately to mind would be, what would happen trying to access or print at an empty slot in a collection: additional code would be required to facilitate such occurrences.

```

get Rd number
if (Rd number != 0)
  go to start of rule list
  while (Rules to be read)
    aRule = next Rule collection position
    while((Rd number != aRule operand) && !EOL)
      get next aRule operand
    endwhile
    if (aRule operand == Rd number)
      mark aRule for deletion
      identify aRule condition number
      if (only path to Condition No)
        while (Rules to be read)
          bRule = Rule collection at position aRule
          while((condition number != bRule operand) && !EOL)
            get next bRule operand
            if (bRule operand == condition number)
              mark bRule for deletion
              if (only path to Condition No)
                condition number = bRule condition number
              endif
            endif
          endwhile
        endwhile
      endif
    endif
  endwhile
endwhile
end

```

Figure 3.8: Deleting Riskdrivers

The collection compressing idea gives the collection a lot more flexibility

i.e. the code for accessing and displaying collections of objects can remain consistent with the approach adapted for other Objective-C systems. Because of the classes and methods provided by the Objective-C environment, the compression and updating of collections is greatly simplified. The collection methods which handled deletion, adding and sequencing could be used to reduce the complexity of this problem to an acceptable level as shown in figure 3.8.

After entering a numeric value inside the riskdriver collection range, the Rule collection is searched for Rules, which will be affected by the deletion of that riskdriver. If a Rule operand is associated with a deleted riskdriver, then this Rule is also marked for deletion. The Rule's condition number may also be marked for deletion, if this Rule is the only link to that condition number (i.e. if the Rule object is being deleted, then the text object and the Rule with the corresponding condition number may also have to be deleted). An additional search for Rules containing this condition number must also be made (i.e. Rules containing this condition number must also be deleted). This process is iterated until the condition number, associated with a Rule cannot be triggered by another combination of Rules or the list is exhausted. This operation must be performed for each Rule in the collection. The Objective-C source code for this operation is shown in the "delete" method in the Risk class Appendix D.

This section of code relied heavily on the power of the Object Oriented approach and the Objective-C environment. Without inheriting the facilities of the foundation classes, the work involved in removing and resizing these collections would have been substantial.

- **Amend** Facility for the SuperUser to change any of the riskdriver values presented to the user. While amend can be considered a factory operation common to all riskdrivers, the changes are made to individual instances of the Class Risk. Using methods in the ordered collection (OrdCltn) and its inherited collection class (Cltn), individual instances can be retrieved for interrogation. By storing all the riskdrivers in an OrdCltn instance called riskCltn, access to any riskdriver may be achieved with statements such as.

```
aRisk = [riskCltn at:4]
```

Here the fifth object in the riskCltn is returned to the object label aRisk. All collection counts start at zero, hence the fifth object being retrieved when four was specified in the statement. Ensuring that the riskCltn contains only riskdriver instances is verified by the add: method in the Risk class. This method will verify that only instances of the Risk classes are added to the risk collection. Checks

also have to be included, to ensure that only valid collection slots(i.e. slots which contain risk objects) are called, otherwise a run time error will occur.

To prevent data inside an object being wrongly amended or lost during any amend session, a copy of the instance is made for amending. Changes during the amend session are made to this dummy object instead of the original selected instance. Some of the Objective-C statements required to perform such operations are listed below. The dummy object reference here is created by the Risk class with the method "new" instead of "create" so that the template will contain nil values. The object marked for amending then copies across its instance variables to the dummy object.

```
aRisk = [riskCltn at:--num]; // retrieve instance
dummy = [Risk new]; // create dummy instance
[dummy copy:aRisk]; // copy instance variables to dummy object

[dummy maintenance:num]; // perform necessary amendments
                        // to dummy object

// if changes are OK
[[riskCltn insert:dummy before:aRisk] remove:aRisk];
```

This approach was the safest way to change the object's private data, performing copy operations on the objects as below.

```
strcpy(riskdriver,[aRisk riskdriver]) ;
strcpy(risktxt,[aRisk risktxt]) ;
```

The riskdriver variable on the left belongs to the dummy object which called the "copy" method (i.e. dummy->riskdriver represents riskdriver). The method [aRisk riskdriver] returns the riskdriver title string to be copied. Only when the amendments to the dummy object are correct, will this object be allowed to replace the original. The dummy object replaces aRisk in the riskCltn using the insert:before: method.

The Objective-C language does provide an alternative mechanism, besides objects, for accessing an objects instance variables. This is one of the strengths of the language, but its also one of its greatest weakness because it violates the principles of encapsulation. Direct addressing of an objects instance variables is permitted, but such operations can be dangerous and are not recommended by the suppliers.

• **View and List** The SuperUser of the tool also needed the capabilities to be able to view and list without affecting their contents. With dynamic collections, it is important to be able to print any or all of the objects in collections of various sizes. It is possible to write Objective-C code which will overcome these constraints. The following three lines of code, highlight the ease and simplicity of printing a collection of unknown size and contents using this approach. This code will remain unchanged no matter what objects are added to the collection. The only constraint on the object added to `objectCltn` is that it knows how to print itself.

```
objectSeq = [objectCltn eachElement];  
while (anObject = [objectSeq next])  
    [anObject print];
```

Printing the riskdriver collection is simply a matter of replacing `objectCltn` with `riskCltn`.

3.4.2 The Project Manager

Most of the chapter until now has been focused on the functionality required by the SuperUser. The SuperUser uses these operations for setting up an environment, which would allow the tool to analyse project risk. However the real user (the person whom the tool is designed for) of the Risk Analysis tool is the project manager. Project managers need a tool which will permit comparisons of their project characteristics against "Rules and Risk Areas" which they presume have been entered previously by some expert(s) in software management. The operations which the project manager needs to perform are described by the following areas of functionality.

- [1] View Riskdriver
- [2] View Rule
- [3] Amend Riskdriver value
- [4] Amend All Riskdriver values
- [5] Print Risk Report

View riskdriver Each riskdriver in the tool represents an area of risk which is related to software development. Allowing the project manager to view various risk areas, means that the risk instance must be retrieved from the risk collection object "riskCltn" and the IS database must be interrogated to find the value of the riskcondition chosen to represent the project characteristics for this riskdriver. The diagram in figure 3.9 illustrates the flow of data and the software entities involved in viewing riskdrivers.

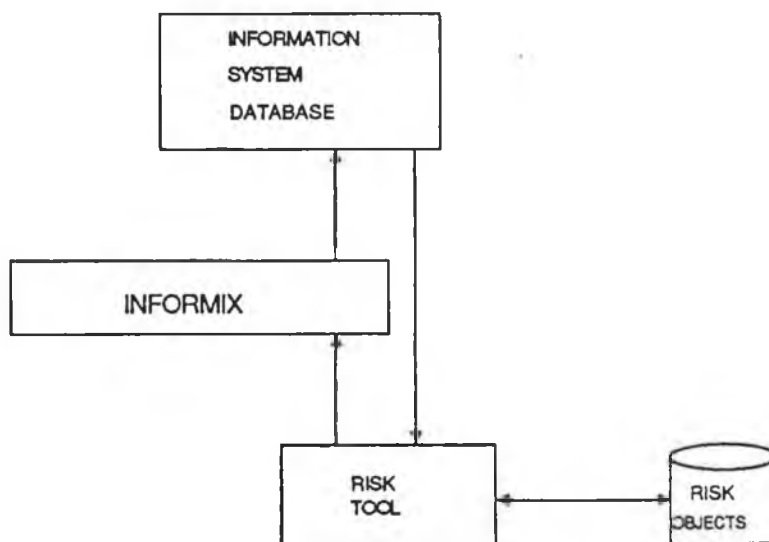


Figure 3.9: Getting risk values from the database

The IS database contains all the relevant information associated with the workbench. Project characteristics elicited from the risk analysis tool are included in these tables. When the project manager asks the Risk Analysis tool to view a riskdriver, all the characteristics related to the riskdriver may be retrieved from the riskCltn. However the riskcondition value, representing the project is stored in the IS database. The Risk Analysis tool sends a message in the form of an Informix statement to the IS. If Informix validates this statement as correct, the row and column in the appropriate database table is found and the value is transferred to a method for displaying the riskdriver value. Most of the problems associated with this functionality were related to calls to the IS database. Because the database was developed at a different geographical site, compatibility problems existed when integrating the tool into the workbench. The code for retrieving riskdrivers from the riskCltn is inherited from the collection classes and is easy to implement. However retrieving the Risk object values from the database is completely different, requiring the new methods for interrogating

the database.

View Rule Gives the project manager the ability to view the Rules which are applied to riskdrivers for a project. The representation of Rule operands eg. R1.1, R2.1 etc, will not be comprehensible to the project manager starting to use this tool. Therefore is important that when the project manager views Rules as they are displayed in standard english format.

When the project manager wishes to view the contents of a Rule the first thing to be determined is which Rule? The project manager must provide some input which will determine Rule instances. Going back to the example Rule collection in figure 3.3, assume the project manager wishes to view the Rule(s) which trigger the diagnostic text associated with Rule (condition number) five.

R7.2		*						-> C5
R7.1		R8.1		*				-> C5

Figure 3.10: Rule operands for Rule Number five

From the table it can be seen that two instances of the Rule class can be related to this condition number five. The values of Rule operands on the same Rule represent an accumulation of riskdriver values which must be true to trigger this risk condition. The operator AND was invented for the insertion of Rule objects so that instance could have more than one operand. The number of Rule instances relating to any condition number will depend solely on the Rule syntax entered previously by the SuperUser. The OR signifies a different Rule instance pointing the same Rule condition as some other instance.

By reversing the procedures used for validating Rule operators and operands, it was possible to replay the values enter by the SuperUser back to the project manager. By setting up two queues, one for operators and one for operands the Rule information is transfered from it's storage representation above, to the appropriate queues. Reading along the Rule operands, we enter AND between operands, and use OR to represent a new instance of Rule which has the same condition number. The end of references to a particular condition number(rule) is indicated by THEN in the operator. The resulting queues are shown in figure 3.11.

R8.1	THEN
-----	-----
R7.1	AND
-----	-----
R7.2	OR
-----	-----
operand	operator

IF R7.2 OR R7.1 AND R8.1 THEN

Figure 3.11: Operand and Operator queues

By further expanding the risktext and riskconditions for each riskdriver the presentation of the Rule should be in english format as below. All lower case text refers to operands, upper case refers to the operators. By reading from both queues alternatively it is possible to reconstruct a Rule sentence.

IF in relation to what we are accustomed the product is -
 very small or is easily broken down into normal size work
 packages OR in relation to what we are accustomed the product
 is -
 fairly small or fairly easily broken down into normal size work
 packages AND in relation to what we are accustomed - require-
 ments
 are very simply and easily allocated to software
 components/modules THEN

Amending riskdrivers This is the project managers only input into the tool which will affect riskdriver values stored in the IS. This is to reflect any changes in project characteristics, or invalid input entered by the project manager. The section on listing riskdrivers discussed retrieving project values from the database. A similar operation must be performed for amending riskdrivers only this time it must be possible to update the IS table. The operation required to perform amendments is listed below.

Determine which riskdriver to amend;
Retrieve its associated value from the IS database;
Display all the information to the project manager;
Allow changes to be made and confirmed;
Update the IS database with the new riskdriver value;

Amending all riskdrivers use the same operations, the only difference being that the amend operations are performed sequentially on all riskdrivers in the riskCltn.

Printing Results As mentioned earlier, the project report from the Risk Analysis tool is divided into two sections.

- [1] Percentage risk values for the four risk management areas
- [2] Generating the appropriate text related to a risk area

The functionality required to provide this output was described in Measure and Text classes. The presentation of the output generated from these classes to the project manager is vital for the future success of the tool (i.e. if the output is complicated and unstructured, the project manager can be put off using the tool).

The percentage risk in the four risk management areas needs to be printed to a project report and to the VDU. The textual information is also sent to the project report and to the VDU, however the details printed at both mediums will be different. The diagnostic text relating to all Rules in a "true state" are sent to a printed report. Only the top level predicate conditions display explanatory text to the VDU, describing what is making the project risky. Figure 3.12 shows a tree of related Rules set-up by the SuperUser.

If all the Rules shown are true, all diagnostic messages associated with condition clauses are sent to the printer report. However only the explanation text associated with C4 needs to be displayed on the screen i.e. the fact that diagnostic message C4 is being displayed means that C1 and C3 must be true. The tool also provides a mechanism for tracing back along this Rule tree.

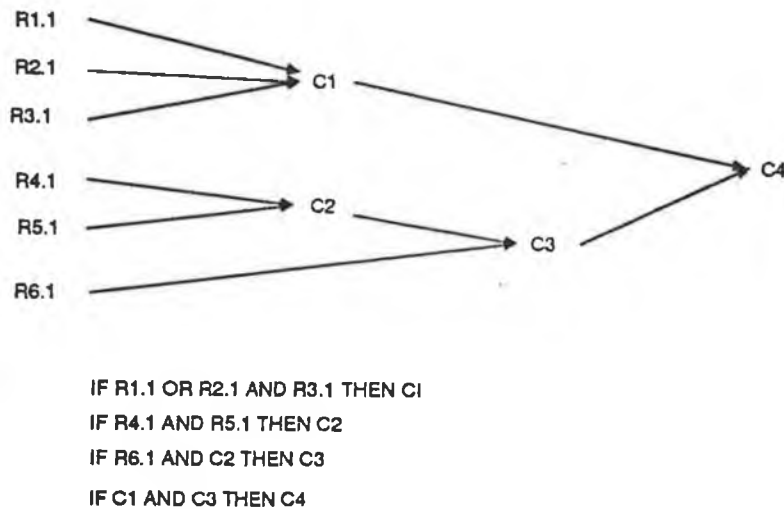


Figure 3.12: Rule Tree

Rule Interrogation If this option is selected, the second section of the Risk Report is displayed. The manager may select any Advice Paragraph (with the mouse) with a view to discovering why it appears in the report. The corresponding “Why?” paragraph is then displayed to provide explanation. If the manager wants to “dig deeper”, he/she can select the “Why?” paragraph. The tool will then, by backward chaining, use the pointers stored away by the “Compute Risk Report” function, display the “Why?” paragraphs associated with successful rules upon which the selected “Why?” paragraph is directly predicated ...and so on right back to the values of the individual Risk Drivers if desired.

3.5 Summary

The Risk Tool works at a fairly high level of abstraction. It deals with broad issues relating to the project as a whole. It would be valuable to supplement Risk tool with a series of more detailed risk models relevant to specific phases of the project lifecycle. These models could be called on, as the project progresses.

The scope of the current “risk management areas” is rather limited. In particular, the current risk drivers focus largely on “internal” sources of risk to the project. There is no coverage of risks arising from, for example, any contractual/legal aspects of the project.

Chapter 4

Calendar Tool

4.1 Introduction

Associated with the success and failure of all projects is the time duration required for the completion of the task. Time plays an important role in the successful completion of tasks and subtasks. Whether the project we are working on is building a house or building a computer system to control a country's finance for the next ten years, the project must be carefully planned and completed inside a fixed time scale. In large projects tasks are broken down into lower level subtasks each of which must be completed by a certain date, before work can commence on other sub tasks required to achieve the overall completion date. Throughout these projects meetings occur to review progress, checking on the resources applied and the products produced from these tasks. The Calendar tool described in this chapter provides this information for the user (project manager) of the IMPW, by retrieving the required project task information from the IS database.

It was decided at the outset that the Calendar tool would only be used for the retrieval of information. This would eliminate the possibility of amending values created by one of the other tools. A decision was also made early in development as to what type of information should be retrieved and displayed by this tool. Section 4.3 will give a detailed description of the information presented by this tool. This information will contain dates of events, activities and milestones which are important throughout the lifecycle of software development. Whether or not current development is in tune with these dates will be of no concern to the Calendar tool. Slippage in the date of task completion, due to late delivery of some product or lack of resources etc, is adjusted by other tools in the IMPW.

This chapter also looks at the main classes which were created for the development of the Calendar tool, showing how these classes fit in to the Objective-C inheritance mechanism.

4.2 Calendar Overview

The Calendar tool acts as a project clock which allows the project manager to view aspects of progress information collected by other IMPW tools. The manager may view this progress information at varying levels of detail, corresponding to different levels of resource monitoring. The tool offers three operation modes for viewing project details.

- [1] Open
- [2] Interval
- [3] Task

The information which is displayed by the Calendar tool is acquired from the database tables in the IS. As the database contains a large amount of project information, deciding what information to display and in what quantity was one of the earliest problems encountered while developing this tool. Unlike Smalltalk-80 which has a library class dedicated to the "Date", no similar operations are provided in the Objective-C foundation library.

The development of the Calendar tool therefore involved the creation of several class objects, which were attached to the Objective-C inheritance mechanism as shown in figure 4.1.

The state objects(CalInitialState, CalSecondState, CalThirdState) and CalAutomata are used for setting up a finite state machine for the Calendar tool. The CALLoad class was created for retrieving project details from the IS database. The various sub tasks associated with the completion of a task being viewed by the Calendar tool are stored in the Task class.

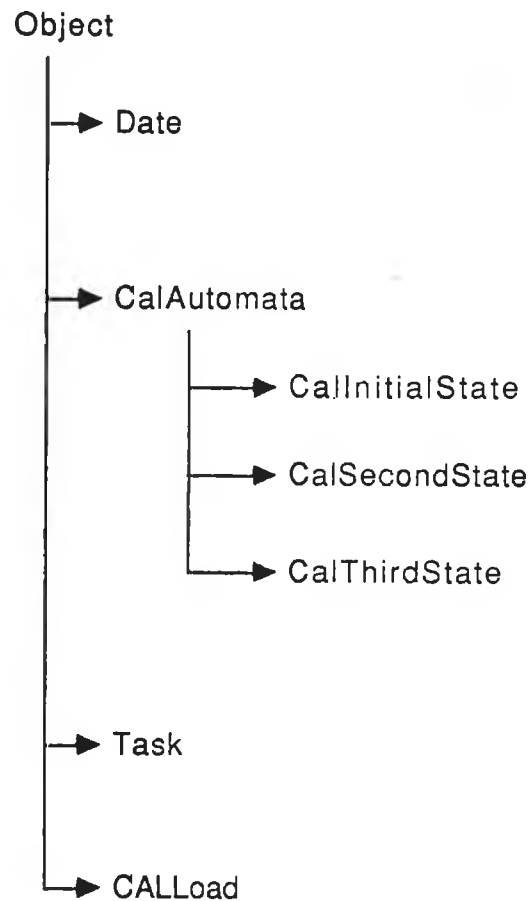


Figure 4.1: Calendar Tool Classes

4.3 Calendar Presentation

The role of the calendar is defined as a tool which supplies information but does not permit updating of this data. Sending calls to the IS database from the CALoad class will enable information relating to projects to be displayed by the Calendar tool. But what information should the project manager see? Should we just blitz the project manager with all data related with the project under scrutiny? Complex software projects will contain vast amounts of detail, this will have repercussions on memory space, speed of operations, the tool interface presented, as well as the complexity and reliability of source code.

The success/failure of this tool like many others which are used solely for the presentation of information will be dependant not only on the details that are presented to the user but also how they are displayed. Deciding what information to display and how, was guided by other tool displays in the workbench and the

literature referenced[Bar 86],[SPE 86],[Cox 86],[Kee 81].

The main goal of the Calendar tool was to present the project events and activities, created by the other workbench tools, which would mark important milestones in project development. It was also envisaged that the Calendar tool would present this information in a summary format and provide some simple method for obtaining more details on highlighted information. Other tools such as the Pert and Gantt perform progress monitoring and resource scheduling of project events and activities. It was important therefore to create a Calendar tool which not only conformed to the goals given above, but did not display output similar to the previously mentioned tools giving the notion of identical tools in the workbench.

The information represented by the Calendar tool selected five areas from the IS database to display important project characteristics which would be helpful to the project manager. The areas selected were imports, products, meetings, work in progress and personnel. Other areas, such as allocation of resources and cost were adequately described in other tools or as part of one of the five sections mentioned, hence their exclusion.

Imports Most projects depend on materials and resources from some external source. The foreman on the building site will import bricks for building, if the project has been planned correctly, they should arrive at some date before the bricklayers. Software projects also depend on materials and resources from outside their project boundary. The implementation of some software tasks may depend on a module of code developed by some external software house, or on the output from one of the other sub tasks in their project. It is important therefore that some indication of imports, required by the task under view should be indicated.

Products The output from the completion of most projects and project sub tasks will be some type of product. In software development these subtasks could be a feasibility study, a coding module, an Input/Output interface etc. Knowing when these sub tasks are completed provides important information to the project manager and hence their inclusion in the calendar display.

Meetings During the life span of any project there will usually be a number of scheduled meetings. The project manager will have numerous meetings associated with the project discussing topics such as costs, design, resources, quality etc. Highlighting to the project manager, the dates of these meetings was considered a desirable feature.

Work in Progress During the various phases of project development different tasks will be performed concurrently, in an effort to speed up the completion of the project. Knowing when work on tasks is been carried out over any particular time period, plays a significant role in the allocation of resources. This facility also indicates the nature of the work being carried out during a certain time period when required.

Personnel During the life of the project there may be numerous people involved in the completion of the various tasks. Project managers need to be aware of what people will be involved with certain tasks on certain dates. The personnel facility was included to indicate when personnel are involved with the task and also what personnel are involved.

To show all the related data for each task event required for the completion of a medium to large project would cause a great deal of congestion on the calendar output display (i.e. the number of personnel involved could be in the hundreds). This problem is eliminated by only displaying some type of indication if a particular event has occurred on the respective date instance being viewed. The calendar view displayed in figure 4.8 uses asterisks to represent an event occurring on a particular date. Further information on any of the five events highlighted by asterisks can be obtained using a selection mechanism, which gives a more detailed description of the selected activities for that date range.

4.4 Date Class

The main object in the Calendar tool is the Date factory class. The instance variables in this class represent the date and the events if any, occurring on that date. Boolean values were used to signal the activation of the respective events for date instances and the collection of events stored the sub-tasks which actually occurred in the duration represented by the date object. The data structure for defining instances is shown figure 4.2.

All instance variables are important to an object, otherwise why should they be defined in the first place? However the instance variable `day_number` can be regarded as the principle variable in the Date class. It is the value stored in this variable which determines the data values related to the other variables.

There were a number of ways of allowing a date instance to represent the calendar date i.e. `string`, `integer` or C `struct` combinations. The data type chosen however for the Calendar tool was the C type `long`, the reasons for choos-

```
= Date : Object (CalGroup, Primitive)
{
    Instance variables
    long day_number;    // days since Jan 1st 1970
    BOOL day_imports;
    BOOL day_products;
    BOOL day_meetings;
    BOOL day_personnel;
    BOOL day_work_in_progress;
    id anImportCltn;
    id aProductCltn;
    id aProgressCltn;
    id aMeetingsCltn;
    id aPersonnelCltn;
}
```

Figure 4.2: Date Classes Instance Variables

ing this type were influenced by two main factors. If the project managers view of the tool required changing so that individual dates could be viewed on hours, minutes and seconds, it would be beneficial to have a data structure for the class that could facilitate this request and would not require substantial changing. The second reason was, because the Unix operating system which provided the run-time support between the IMPW tools developed in Objective-C and the computer hardware, also stored date and time values as type long. Using this representation it is possible to reference date by calendar day, hour, minute and second. Each increment of one to the long value storing the date represents the addition of one second to the date.

4.4.1 Storing the Date

Representing the date as a long type can be justified for storing date values in the Calendar tool. However asking the project manager to enter the date in the long format introduces a number of problems as shown below.

Enter date :- 347563202413

Project managers entering the date in this format would have the difficult and error prone job of converting the date to a long value. This approach also gives a poor representation of the date types to which a project manager would be accustomed. To ensure continued use of the tool, a more understandable and easy to enter format had to be provided for the project manager. This prompted the writing of a method for the Calendar tool, which allowed the date to be entered in a more conventional manner which could be understood by the user(i.e. May 21 1989).

Once a valid date is entered, the date is converted to a variable of type **long**, so that it may be understood by the Calendar tool. Validation of the project managers input is performed to ensure the creation of correct instances. As with the creation of rules in the previous chapter, the creation of the new object is postponed until the object is validated. Once validated the date could be applied to the formula below for the creation of a new date instance.

```
long_date = (year * 3153600) + (daysBefore_month * 86400) +  
            (day * 86400) + (leapyear * 86400)  
  
year = 1989 - 1970  
daysBefore_month = may = 121  
day = 20
```

Figure 4.3: Date Conversion Formula

The date range permitted by the tool is any date from January 1st 1970, hence the reason for subtracting 1970 from year. The daysBefore_month value is the number of days from the start of the year to the start of the current month. In the example above one hundred and twenty one days have occurred since the start of the date month.

The variable day represent the number of days into the month and leapyear contains the number of extra days to be added, due to leap years since January 1970.

The numeric value 3153600 represents the number of seconds in a normal year (eg. one which is not a leap year). The value 86400 represents the number

of seconds in a day. After the `long_date` has been calculated it is passed as an argument to create a date instance as below.

```
aDate = [Date create:long_date]
```

The effort required for entering input to the tool can be further reduced by accepting the current system date as the default date, in response to pressing the enter key, when prompted for the date. Using the Unix operating system call `gettimeofday()` [Sun 86], it is possible to obtain the current date from the computer. Although the `gettimeofday()` function is not part of the Objective-C machinery the Object Oriented principles for reusable code can be applied. If the computer validates and uses this function, then there is no need for the operation to be repeated in the Calendar tool source.

When an instance of date is created, all the necessary task components described earlier must be associated with the import, product, progress, meeting and personnel collections for that date instance. The retrieval of this information is explained in the `CALLoad` class.

4.4.2 The Date Collection

Once the date instance has been verified and the time interval each instance has to represent is established, the Calendar tool must generate a number of date instances which are added in an ordered sequence to a collection of date objects. Before creating this sequence of date instances, a decision had to be taken on the number of instances to be created and stored in computer main memory during tool implementation. Should instances representing dates form the start to the end of the project get created to represent an entire project? In large projects (i.e. those lasting three or more years), trying to store all events on a daily basis would cause memory problems. Another alternative (the one adapted for this tool) was to create a certain number of instances after the initial date, to represent the task events.

The number of instances created after the initial date has been set to thirty one for this version of the Calendar tool. The reason for selecting this number was simply to allow project managers to view the tasks on a daily basis for an entire month. If the project is large and complex then the interval mode should be used to represent the thirty one instances with a more correct scale. Setting up a routine to handle thirty one date objects is simplified by inheritance which uses classes and methods provided by the Objective-C foundation library as below.

```
printf(dateStr,"%d-%d-%d",[aDate year],[aDate month]
[aDate day]);

dateStr = "1989-10-22"
printf(dateStr,"%s %d %s %d",[bDate dayName], [bdate day],
[bDate monthName], [bDate year]);
```

```
dateStr = Thursday 22 October 1989
```

This section has mentioned only some of the methods developed for the Date class. Additional methods for comparisons, interrogation and presenting the date can be viewed in the Date class definition file Appendix E.

4.5 CALLoad class

Once the date instances required to show the project have been established, it is important to have a tool interface which responds quickly to interactions made by the project manager. Reading and writing from the data base will be time consuming no matter how efficient the searching technique. Some date instances may contain large amounts of information extracted from the IS, while other instances may contain no information at all.

Searching for information relating only to the five date instances being displayed, may improve the time to set-up the initial display. However if we wish to alter the dates presented by the tool, the tool mechanism must grind to a halt until the new date instances have the required information, which is stored in the IS, linked to their respective collections. For the project manager who wishes to scan backwards and forwards through the date collection, this technique is undesirable, because of the abundance of memory provided by computer workstations and the fact that only one tool at the time can be implemented in the prototype version of IMPW. It was possible to read from the database all the events associated with a specific task. The retrieval of information from the IS might be longer than in the previously mentioned method, however once all the details have been loaded into memory the project manager will be able to scan across dates, without having lengthy waits for additional data to be load from the IS.

The loading of events related to the project task being viewed by the Calendar tool is initiated by the following Objective-C statements.

```
aCALLoad = [CALLoad new];  
[aCALLoad loadDM];
```

The CALLoad class defines one instance variable the object taskCltn which is aggregated to the isa pointer inherited from the objects super class Object.

This instance variable will contain a collection of all sub tasks associated with the completion of the top level task. After creating the instance to hold the sub-tasks, the method loadDM triggers the retrieval of sub-tasks related to this top level task from the database using the command.

```
aCommand = [[[String new]
concat:[String str:"SELECT Name,EarliestStartDate,
EarliestEndDate"]
concat:[String str:" FROM Task"]]];
```

This retrieval of tasks from the IS is made by sending Informix commands to the data base. The concatenation of the Objective-C objects above, help create an Informix command for retrieving all the sub-tasks related to task. After determining all the sub-tasks, the imports, products and resources associated with these sub-tasks must also be retrieved from the IS. The method retrieve:and:from: in the CALLoad class restores for each sub-task the appropriate import, products and work information.

4.6 Task Class

The entity Task in the IS represents all the work involved in completing a project. Each task or sub task involved in project development can be viewed as an individual project. The task object contains a collection of sub tasks, the sum of which result in the completion of the top level task. All imports, products and work operations which occur during the duration of the task must be related to the respective tasks.

The class Task was created to store all the necessary data associated with sub tasks from the database. Figure 4.4 shows the instance variables defined in the class definition file.

The id name is required to identify the name of the sub task, the (id) objects startdate and enddate use the String class to help represent the duration of a task. The product object will contain when required, a collection of products for this sub class. The consumedCltn will contain a collection of all the components that will be consumed by the task. The workCltn will contain a collection of work resources associated with a task.

After obtaining all the task details, the object is added to the taskCltn. The Date class then relates events to date instances by searching through taskCltn.

```
= Task : Object (CalGroup, Collection, Primitive)
{   Instance Variables
    id name;          // task name
    id startdate;
    id enddate;
    id product;
    id consumedCltn;
    id workCltn;
}
```

Figure 4.4: Task Class Instance Variables

Five different searches are made for the different collections defined in the Date definition file. Start and end dates of task in the taskCltn are compared against the date stored by the date object. When a valid date is obtained, the respective event collection is added to the instance variable collection which is related to the search.

4.7 CalAutomata and State classes

The CalAutomata and State classes although not linked through inheritance, are used together to create a finite state machine used for controlling the operation paths while using the Calendar tool. The tool could have been developed without such an environment, but because the integration was been implemented at a different geographical location, the approach helps to minimise and control changes. The diagram in figure 4.5 shows the various states created by the automata machinery for the Calendar tool.

4.8 Operation modes

Because the tool is used only for displaying information, the amount of functionality regarding tool use, is greatly reduced. Indeed the only functionality required, refers to the way in which date objects should be displayed. Having a

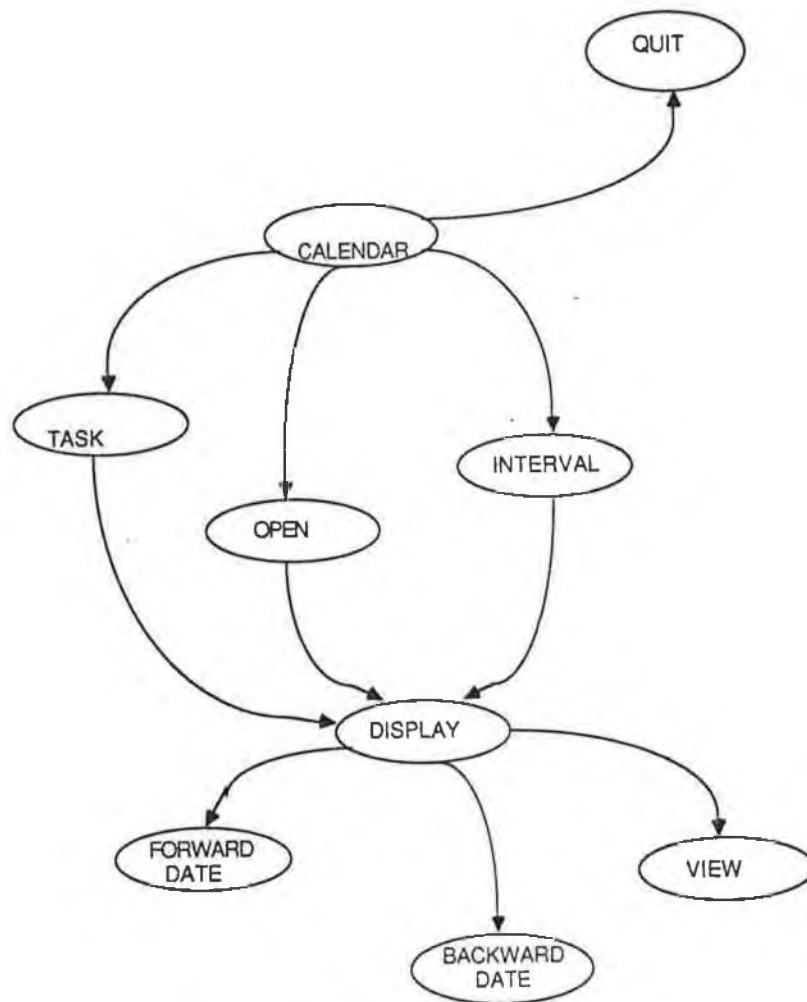


Figure 4.5: Calendar Automata

tool which only permitted viewing on a daily scale may cause many date instance to appear with duplicate information in large projects. Functionality was added to the tool, to allow the project manager to determine, the time period each date instance should represent and to permit a calendar view of ordered tasks. The different views of calendar dates is made on entry to the tool, where the project manager is confronted with the following options.

1. Open <defaults to one day intervals between dates>
2. Interval <the project manager specifies intervals for date instances>
3. Task <the interval between calendar dates is determined by project tasks>

4.8.1 Open and Interval Modes

The only difference between open and interval modes is, that interval modes permit the project manager to adjust the time period between date instances. The Objective-C code required to create date instances is used for both modes of operation. The only difference is the value of the argument `datescale` which determines the date interval span. The sequence of date instances are created by incrementing the `long_date` value as below.

```
long_date += datescale;
```

In the open mode the `datescale` is set to 86400 so that each new instance created for the date collection is a day greater than the previous instance. When in interval mode, the project manager enters an integer value to represent the time span for each date instance. This integer is multiplied to `datescale` before it is used in the formula given above eg. `datescale = 86400 * 7` causes an interval of one week between instances as shown in the main window view in figure 4.6.

Date	1989-06-14	1989-06-21	1989-06-28	1989-07-05
IMPORTS	***		***	***
PRODUCTS	***		***	
MEETINGS	***		***	***
PERSONNEL	***	***		
WORK PROG		***		

Figure 4.6: Calendar View

4.8.2 Task Mode

When in task mode, the date instances in the calendar display are sequenced in event order. The taskCltn which contains all the sub tasks associated with the completion of a particular task, is sequentially searched to find the respective starting dates for each sub task. The starting date instance variable in each task object is of type object(id). This decision was made to utilise the code provided in the foundation collection classes. Objective-C provides tried and tested methods for sorting and storing objects inside collections. Date objects for the tasks which are stored in an ordered collection(taskCltn) are being transferred to a sort collection(SortCltn). The dates representing the events are sequenced in their correct order and stored in a sorted collection object of type SortCltn.

Objective-C as mentioned provides a foundation class for storing sorted collections "SortCltn". This collection allows collections to remain sorted at all times and for insertions to be added at their proper place. However when objects are being added to a SortCltn, some type of comparison must take place between objects. This in turn implies that the sorting class SortCltn knows in advance(before run time) the objects types being sorted. This would contradict the dynamic binding principle advertised by Objective-C.

4.9 Event Details

For SortCltn to qualify as a true Objective-C class, the sorting of various object types should be permitted without amending existing code. The code below describes the creation of an instance of SortCltn. Note the method required to do the object comparisons is passed as an argument.

```
anEventDataCltn = [SortCltn orderBy:"taskcompare:" onDups:1]
```

The "orderBy:onDups:" method refers to a specific method "taskcompare:" which knows how to do the comparisons, for the objects being stored in the sorted collection in date order. It is the responsibility of the person who stores the objects in the sorted collection, to write the method which does the comparisons on these objects. In the Calendar tool instance comparisons are performed by the method "taskcompare:" which compares the start date of the objects stored in the task collection(taskCltn). The onDup:1 is used to stop duplicate values begin added to the sorted collection(i.e. the Calendar tool should not see the same date object displayed twice).

Tool : CALENDAR Project : IMPW

From : Thursday March 2 1989

To : Thursday March 2 1989

Work in Progress :

dev syst A

Software Development

Software Realisation

Code Software

YES NO

Tool : CALENDAR		Project : IMPW			
	1989-2-28	1989-3-1	1989-3-2	1989-3-3	1989-3-4
Imports		***	***		
Products		***	***	***	
Meetings					
Persannel					
Work in progress	***	***	***	***	***
Quit					

Figure 4.7: Calendar Interrogation

The information displayed by the three operation modes only provides an indication of the occurrence of particular events. The Calendar tool however allows the project manager to obtain more detail on events occurring on date instances. The diagrams in figure 4.7 show the displays developed to achieve these goals. Selection of events was made using the computer mouse (a detailed description of the construction of these graphic displays will be given in the next chapter).

Cells containing asterisks were used to represent events occurring on date instances. In the example shown in figure 4.7 the work in progress event for the date instance representing the 2nd March 1989 was selected. The second window displays the work in progress events occurring for that date instance. The tasks associated with the event also provided selection in their output. This was to allow detailed descriptions of sub-tasks associated with an event.

4.10 Summary

Most of the goals set initially for the development of this tool were achieved. The information relating to events is retrieved from the database and displayed in a comprehensible and correct manner. However the real test for the success/failure of this tool will be dependent on other users. Decisions such as creating thirty one date instances after the initial valid date is entered and displaying only five date instances in the snapshot of the project shown, were influenced by personal taste as well as associated literature.

Changes can be made to the classes which will cause minimal effect on other classes created for the Calendar tool. However work associated with a second version of this tool, may be better directed towards improving the time taken and approach for the retrieval of events. This task would involve not only rewriting of retrieval methods but also the restructuring of the IS architecture.

Chapter 5

The Workbench Interfaces

5.1 Introduction

The workbench consists of a number of discrete tools, which although linked by the IS data base generally work independently of each other. Each tool corresponds to a particular part of the project management process, where the tool consists of a set of closely related functions. Development of the tools used by the IMPW were carried out in three phases.

Phase one Tools were developed in isolation, input and output from the tools was in textual form.

Phase two Tools were integrated into the IS database, the input and output was still in textual form.

Phase three Tools were integrated into the IS in graphic mode.

Phases one and two concentrate mainly on the functionality and integration of tools in the workbench using Objective-C. After determining the functionality and completing integration, effort was pointed towards development of the user interface. This chapter concentrates on the third phase of tool development, discussing the two main areas related to developing the tool interfaces.

- [1] Automata :- finite state machine which controls the operations performed by a tool when in a particular state.
- [2] Windowing System :- extended graphic package that creates and manages the interaction with the tool applications.

The graphic package used for interfacing with the management workbench was supplied by Verilog, one of the project partners working on the IMPW project. The operations and objects which could be created from this package were influential in the way tools displayed and extracted project information.

5.2 Automata

Knowing the number of interactions between the user and any application, makes it possible to setup a finite state machine(Automata) which could simulate these interactions[Ger 82], [Hen 68],[Hop 79],[Min 67]. The finite state machine ensured a smooth transistion between integration phases, as well as providing a navigation system for moving between operation displays.

Interactive sessions for the tools mentioned in the previous chapters go through a series of states, each with a well defined general pattern: A panel is displayed with questions for the user; the user supplies the required answer; the answer is checked for consistency (questions are asked until an acceptable answer is supplied); and the answer is processed.

This generic method 'execute' in figure 5.1 could be used to represent any user interaction state[Mey 87]. Using the power of inheritance and encapsulation provided by Objective-C, it was possible to develop a generic class with abstract methods which could be used by the various subclasses that perform similar operations but use different data.

The diagram in figure 5.2 shows a state graph for the Risk automata simulating all possible user interactions within the tool. Each circle in the diagram represents a risk state waiting for user interaction. The arrows represent the transition of going between states which occur when an event (user interaction) is triggered.

```

- execute {
    do {
        [self display];
        [self read];
        [self correct];
        if ([error number] != NO_ERROR)
            [self errorMessage];
        while ([error number] != NO_ERROR);
        [self treat];
        return self;
    }

```

Figure 5.1: Generic execute method

The circle R0 represents the risk automata in its initial state. After an event has occurred the machine proceeds in a deterministic fashion, that is, its actions in response to a given sequence of inputs are completely predictable. Because of the finite nature, the structure of such machines can be easily used to describe different environments. The work required for describing risk and calendar automata differs only in following state options available(see figure 4.8).

At any given moment, the risk or calendar automata can exist in only one possible state. The next state to be entered is a function of both this present state and the present input. For example, if in the risk automata outlined in figure 5.2, the current state is R1, then entering a value of say one would transfer activities to the R4 state. The next state therefore, always depend upon the previous states as well as the input at that state and so forth, back to the initial operation at R0. Thus the automata may also be viewed as a navigation system, always keeping the interactions within a bounded path. The current state of the machine at any moment serves as a form of memory of past inputs by following the path from the initial to the current state.

The class Automata was used by both tools to describe the possible corresponding states which can be obtained by the tool. The declaration for the risk automata is given below.

Each of the first five instance variables is an instance of a particular state designed to represent that class eg.

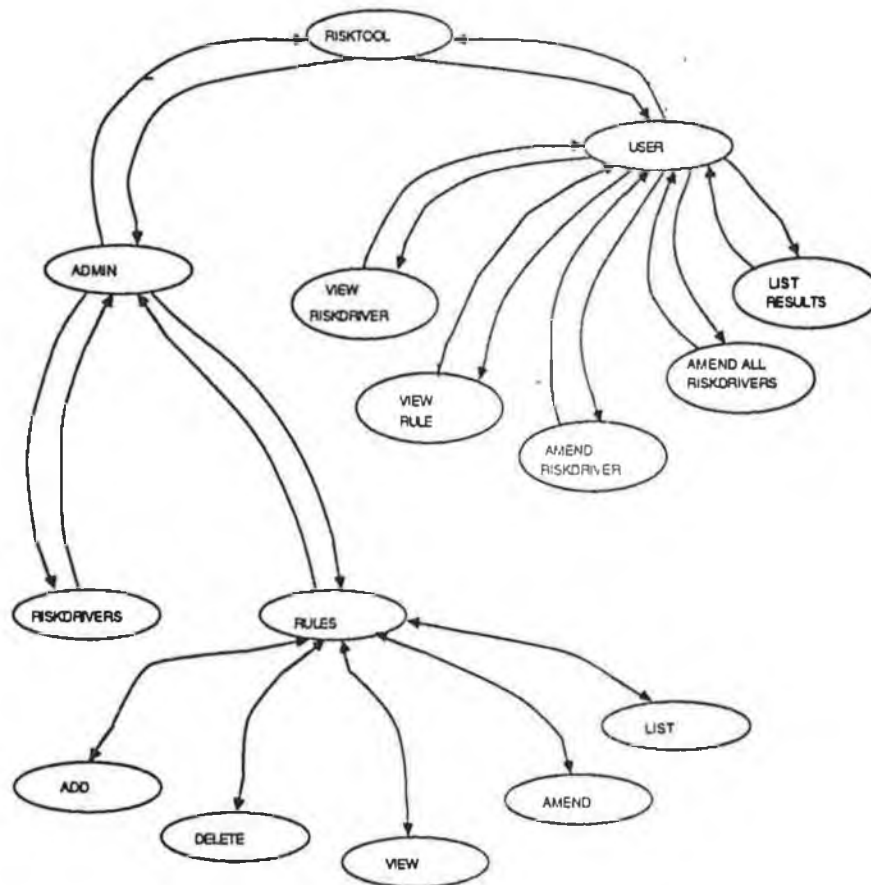


Figure 5.2: Risk Automata

```

firstState = [FirstRiskState new];
secondState = [SecondRiskState new];
    "    "    "
    "    "    "

```

The `currentState` instance variable is used to indicate the state in which the automata is currently executing. The `riskAutomata` class also has an instance variable `automata`, this was included to facilitate multi level finite state machines (i.e. automatas within the automata). This relationship between the automatas is shown in figure 5.3. If an extension from one of the states was required, the new automata can be set-up without affecting the top level automata.

Each state presented by the Risk tool represents a menu display, the `RiskState` classes (`RiskInitialState`, `RiskSecondState`, `RiskThirdState` etc) inherit charac-

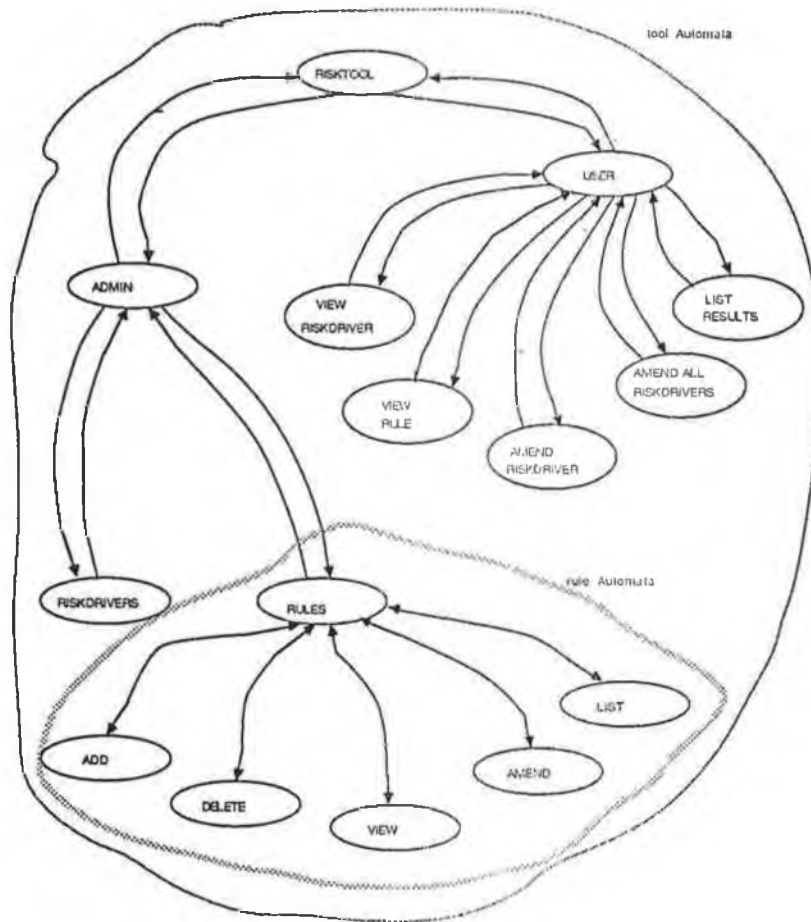


Figure 5.3: Automata within an automata

teristics from the class `State` which contain the instance variables and methods required for implementing the finite state machinery. Each of the `RiskStates` inherits a `stateCollection` instance variable from the `State` class. This variable contains the collection of possible states, which can be reached from the current state as defined by the `riskAutomata`.

The method following `State` in the `State` class works as a central cog in the automata mechanism, ensuring that a correct transition is made from state to state when required.

After establishing the next event from the user, the `stateCollection` offset for that event returns the following state as defined by the automata. The statement

```
[[aFollowingState automata] currentState: aFollowingState];
```



```
= RiskAutomata : Object (RiskGroup, Primitive, Collection)
{ // INSTANCE VARIABLES
  id firstState;    // the initial state of the automata
  id secondState;
  id thirdState;
  id fourthState;
  id fifthState;
  id currentState; // state of the automata being executed
  id automata; // automata in which the automata is contained
}
```

Figure 5.4: Instance Variables for RiskAutomata

returns the current state of the automata. The `currentState` variable for the Risk automata is updated to whatever the following state is.

5.3 Graphic Compatibility

Having created a tool interface in which the user is aware of the current context, presenting only options which are available in that context gives us a correct interface, but it does not solve all user interface problems. As software becomes more powerful and sophisticated, so too are the interfaces used for their representation. The success or failure of many tools in the commercial marketplace today often depend (albeit incorrectly) on the user interface. While the marketing of the IMPW is centered around the functionality, extendibility and malleability of the tools, the importance of a pleasant and easy to use interface could not be overlooked.

The Verilog graphic toolbox which provided a wide range of operations for constructing user interfaces was added to the second phase Objective-C code. The abstract objects supplied by the graphic package included window, box, list, table, network, chart, graph and tree object types. From these objects the interfaces and operations associated with the Risk and Calendar tools were created. While the diversity of such services make the toolbox very flexible for constructing different interfaces, programmers new to the package may find it difficult to use. Discovering the right glue for assembling pieces from the toolbox

```

- followingState {
    int offset;
    id aFollowingState;

    if((stateCollection != nil) && ([event number] != NO_EVENT)) {
        offset = [eventCollection offsetof: event];
        aFollowingState = [stateCollection at: offset];
        [[aFollowingState automata] currentState:aFollowingState];
        [aFollowingState execute: currentObject];
    }
    return self;
}

```

Figure 5.5: followingState method

poses certain technical problems for the first time developer. There was also the problem of using this code inside the Objective-C environment.

The diagram in figure 5.6 shows where the graphic package should ideally be situated, between the application and the interface [Cou 86]. By having three clearly defined areas, changes can be made in one area without affecting the code in the other. This would also support the Object Oriented paradigm of extendible and robust code.

The graphic package however does not permit such a clear cut distinction between these main objects. In order to present tools using windows, boxes etc, from the graphic package, the Objective-C source is interwoven with graphic statements. The diagram in figure 5.7 gives a more realistic representation of the relationship between these objects.

The extent to which the graphic code is intertwined with the Objective-C source will affect the reusability of certain class methods. Further enhancements to this early version of the graphic package may cause changes to methods which reference graphic operations i.e. changing the names of graphic functions, changing the number of arguments accepted by a graphic function, changing the data types required for arguments.

Creating an Objective-C class to contain all the graphic statements and

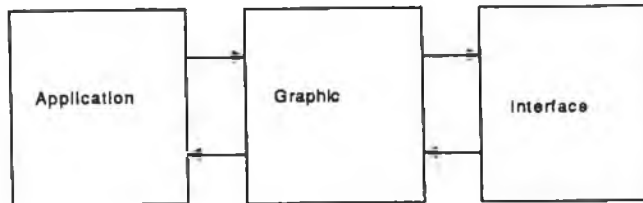


Figure 5.6: Ideal environment for graphic packages

minimise the effect of changes to the graphic package is possible. However such an Objective-C class would have caused extensive changes to the phase two code, which in turn would have caused major problems for the third phase integration. Another problem created by such an approach would be the distribution of operations associated with an object. For example the risk class would have to transfer control to a class outside its hierarchy for printing instance variable information. While possible, this approach is moving away from the encapsulation principle discussed in the first chapter, where only the object that owns the data, should be allowed to perform operations on that data. Because only the display methods for objects required the graphic statements, it was possible to localise these statements within the class definition file. By creating a graphic interface in this way, minimal effort in rewriting tool source may be achieved, while affecting the reusability of as few methods as possible.

5.4 Display Constraints

While designing the user interfaces for individual tools such as the Risk and Calendar, some type of consistency with the interface display of the other tools in the workbench had to be considered. Standards were developed by the different tool builders to enforce some uniformity between the tools. The various tools used different graphic objects from the tool box for presenting the current tool state i.e. network objects are used by the "Work break down" tool to describe the

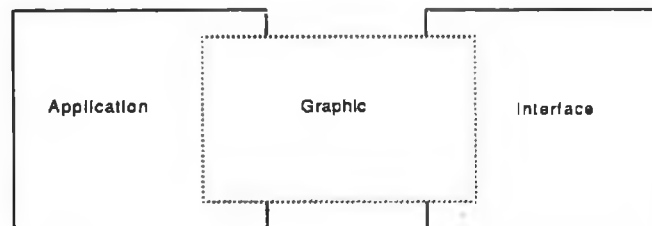


Figure 5.7: Realistic graphic package environment

hierarchical break down of tasks in a project. Other tools such as "Risk" may require different objects such as icocar for displaying textual information. Despite the wide difference in objects used, the physical layout of the windows created, remained as consistent as possible by observing certain constraints. Each window created to represent an interface consists of three parts as shown in figure 5.8.

The top part is used to describe the tool and project name which is currently in use. The middle part contains a working area where the user may interact with the tool while in a particular state. The third part is designated to describing warning messages and confirming user actions. Selected entries for the current tool state are highlighted when required. Highlighting however, was not used to emphasise data, because this may lead to confusion. The physical layout of both Risk and Calendar windows were designed to conform closely with these outlines.

5.4.1 Window Displays

Some of the objects, supplied by the graphic package were more important for describing the Risk and Calendar tool interfaces i.e. windows, boxes and tables, than other graphic objects such as network and charts. These graphic objects such as charts, graphs and trees were considered for presenting Risk and Calendar tool interfaces. Using such objects to create an elaborate interface for the presentation of these tools, was viewed however, as adding confusion rather than comprehension.

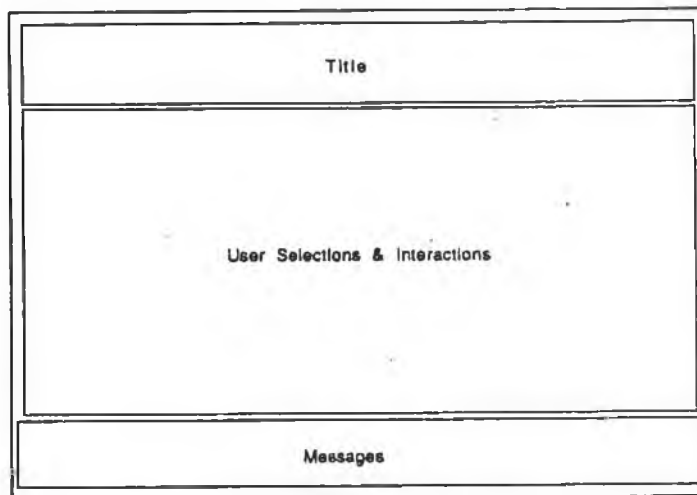


Figure 5.8: Window layout

The Risk tool presents a number of interfaces for interaction with the project manager, all of which contain information to be displayed in a textual format. The presentation of such information using trees objects shown in figure 5.9 was considered an unnatural way of representing riskdriver information.

Using the menu objects and the associated operations was another alternative for viewing riskdriver values. This would have meant displaying all the riskdrivers simultaneously and using push right menus to display the riskconditions. The main problem with this approach is that the character strings for riskdriver and riskcondition values would be too long for displaying on screen (i.e. some are greater than eighty characters).

Keywords could have been used to represent the riskdriver and riskcondition values, with the complete english format of the riskdriver being printed after pressing one of the mouse buttons. However there was also the problem of how to display the risktext and riskhelp information associated with a riskdriver. The idea of using keywords in the menu with a help option for giving a riskdriver description was considered, but like the first method seemed an unnatural way of presenting simple textual information. The layout of the windows finally chosen

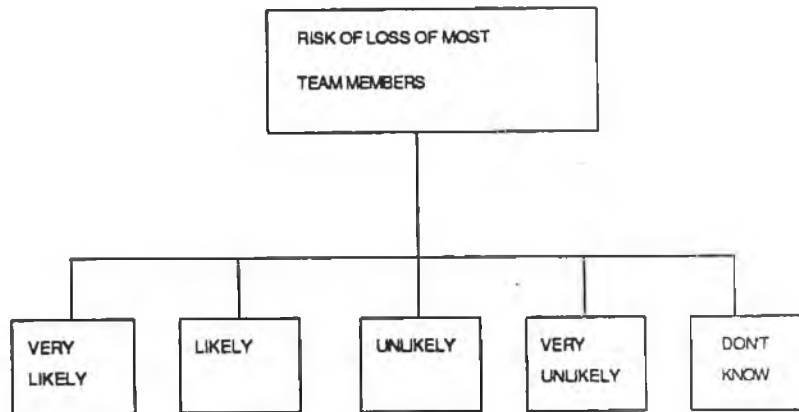


Figure 5.9: Tree presentation

to represent riskdriver displays, used icocar boxes and tables objects is shown in figure 5.10.

The top area in the display is dedicated to the tool name and the identity of the current project. The bottom part is used for quitting from the current state and error messages in accordance with the display constraints discussed earlier. The middle section of the riskdriver window is used to display individual window interactions used by the IMPW tool.

Each box to be displayed in the window view was combined with other boxes to help form the complete picture. The combination of boxes provided by the package is strictly hierarchial. Each box could only have one parent(except the root box which had no parent) and each box could have several children(sub boxes). The boxes defined were one of the following six types.

PROJECT NAME		TOOL NAME	
RISKDRIVER			
RISKTXT		Select Values	
RISKCONDITION	[1]	<input type="checkbox"/>	
RISKCONDITION	[2]	<input type="checkbox"/>	
RISKCONDITION	[3]	<input type="checkbox"/>	
MESSAGES			

Figure 5.10: Final window layout

Box		Description
No.	Type	
1	Empty	box with no contents
2	Input	box contains a prompt and a zone for text input
3	String	box that contains a string of characters
4	Icon	box that contains an icon either as a string of characters or as a picture
5	Graphic	box containing graphic drawings
6	Oneof	box may have any of the representation given above however at any one time only one of these representations is visible

Only by glueing together different box objects offered by the package was it possible to display all the data associate with a riskdriver instance.

5.4.2 Composition of riskdriver window

The composition of all windows used by the Risk Analysis tool included mainly boxes and table objects. Windows displayed by the tools, while being distinct at each state, used similar techniques for displaying and entering information. This helped to reduce the code and provided consistency while using the tool. The operations required for displaying the riskdriver view in the following example were similar to the creation of other windows in the risk and calendar tools. When creating windows to display riskdrivers, the first box to be created was a vertical row box which would be the root box to which other boxes will be attached. Creating the root box from which all other boxes will be related is accomplished with the command.

```
Gevrow_create(arguments)
```

```
risk_rows = Gevrow_create(GEV_VERTICAL,GEV_OUTLINE_ON,  
                           GEV_SPACE_ON)
```

This created the outer box for the riskdriver window on which all other boxes are added. The `GEV_VERTICAL` argument implies, that attaching any boxes to this box will be added in a vertical sequence. The `GEV_OUTLINE_ON` argument implies that the outer boundaries will be displayed. The `GEV_SPACE_ON` is used for the spacing of character and graphic objects.

Having defined the box which forms the outer shell of the risk window display, three more row boxes are added in a vertical sequence. Each of these boxes helps towards presentation of the risk window in a form which provides consistency with the physical layout with the other tool interfaces in the workbench. The first box added represents the tool and project name. The second will display riskdriver details and the bottom box is used for messages and confirming operations. The addition of these three boxes sets up the window skeleton conforming to the constraints set with other IMPW tool builders. The statements required for creating these boxes is similar to that used for their parent box only the new boxes are given unique names and different argument values depending on what objects have to be attached (eg. additional boxes are attached horizontally to the

top box). Once these boxes contain all the necessary details they are attached to their parent box(i.e. risk_rows) with the following commands.

```
Gevrow_add_obj(risk_rows,top);    // top section
Gevrow_add_obj(risk_rows,middle); // middle section
Gevrow_add_obj(risk_rows,bottom); // bottom section
```

5.4.3 Top row details

When using the risk tool for different projects only the name of the project will change. Therefore the code related with this part of the display only has to be called once, when performing operations on different projects. The information displayed in the top section (i.e. tool and project name) requires the creation of two more boxes which are attached to the top box as shown in figure 5.11.

The box on the left hand side represents the tool name. The right hand side box, identifies the project name. The creation of the top box above specified the addition of any further boxes to top box should be added in an horizontal fashion.

Inserting a box with the tool name in a string format is achieved using the icocar option supplied by the graphic package. The icocar box object is a textual icon containing a character string. Issuing the command `Gevicocar_create(arguments)` the tool name was inserted into the left hand side of the top box. The project name was inserted in the right hand side of the top box by attaching two more boxes of type icocar.

Once the information for the top section has been accumulated in the various boxes, it only remains to add the boxes containing the information to their parent, which in this example is the box 'top'.

```
// attach topleft to top
Gevrow_add_obj(top, topleft);
// attach topright to top
Gevrow_add_obj(top, topright);
```

Objective-C being a hybrid language permits the graphic functions to be

```
// Top
// Create the box which the top section of the display
// will be associated with
// box created is a framed horizontal row box "top"
top = Gevrow_create(GEV_HORIZONTAL,GEV_OUTLINE_ON,
                    GEV_OUTLINE_ON);

// Top Left
// Create a framed textual icon
// font size seven, text string is centered
topleft = Gevicocar_create("TOOL : RISK ",
                           GEV_FONT7, GEV_C,
                           GEV_OUTLINE_OFF, GEV_SPACE_ON);

// Top Right
topright = Gevrow_create(GEV_HORIZONTAL,GEV_OUTLINE_OFF,
                         GEV_OUTLINE_ON);
projectTitle = Gevicocar_create("Project : ",
                                GEV_FONT7, GEV_C,
                                GEV_OUTLINE_OFF, GEV_SPACE_ON);
// proj_name is a string variable containing project name
projectName = Gevicocar_create(proj_name,
                                GEV_FONT7, GEV_C,
                                GEV_OUTLINE_OFF, GEV_SPACE_ON);
// the icocars are attached to the topright box
Gevrow_add_obj(topright,projectTitle);
Gevrow_add_obj(topright,projectName);
```

Figure 5.11: Presentation of Tool and Project names

embedded with the Objective-C source. However in trying to make the methods in the various tool classes as reusable and extendible as possible, calls to the graphic toolbox were written inside methods created specifically to relate to the user interface.

5.5 Tool Interaction

The middle area of the window view displays the information specific to the current state of the tool being used. For example if the user is using the risk tool and the tool is at the first state, the display in the middle area will be as displayed in figure 5.12. The current state of the Risk Analysis tool will determine the level of user interaction with the middle window area(eg. interaction with the riskcondition values for any riskdriver is only permitted when the user is using the amend riskdriver mode operation).

Presenting textual information in a form which could be easily highlighted in accordance with mouse operations seemed the most natural way of allowing the user to interact with the tool. Unfortunately when displaying and highlighting selected riskconditions associated with riskdrivers, serious shortcomings with this package became prominent.

The strings used to represent riskdriver title and conditions had variable lengths, going from two characters to well over one hundred. The graphic package provided no text justification, so the appropriate code had to be developed in Objective-C. The fact that Objective-C supplied no code for text justification suggests that perhaps additional functionality should be added to the foundation library in the form of a class or method.

The text justification function written for the tool was added to the Risk class, because this was the only workbench class which required these operations. To follow the principles of the Object Oriented approach the text justification functionality should have been written as a reusable class for use of all classes.

The text justification function sub-divided large strings into separate icocar box types. The accumulation of these boxes was used to represent the text on separate lines. Unfortunately this approach ruled out any possibility of highlighting individual riskconditions. Using combinations of sub boxes to describe textual information, caused undesirable side effects for highlighting the riskconditions relating to the current project characteristics. Presentation of selectable choices is made from a table object, by attaching all the selectable boxes onto the table.

However selectable riskconditions are made up from the accumulation of boxes, therefore making it impossible to select items on individual riskcondition boxes.

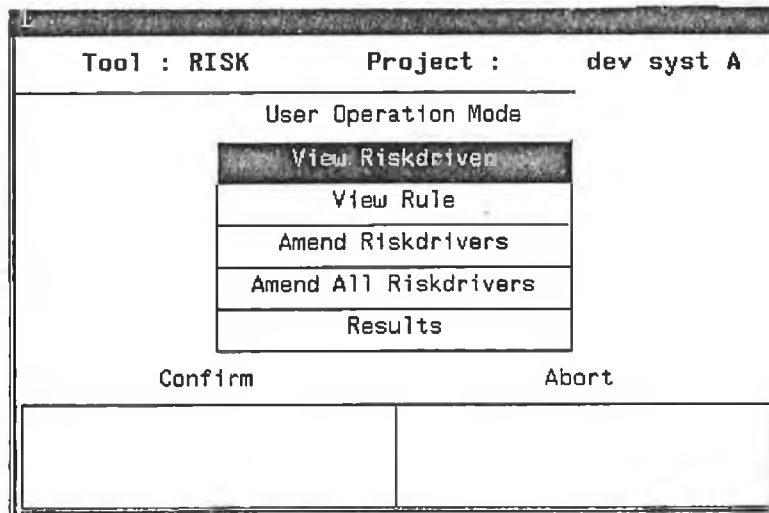


Figure 5.12: Initial Risk Window

The solution for this problem was to add a table object alongside the riskconditions, highlighting the cell opposite the riskcondition value which represented current project characteristics. This solution involved the creation of row, space and table boxes plus numerous graphic function calls to get the desired window display shown in figure 5.13.

5.5.1 Using the mouse

Each area of functionality described for the project manager in chapter four required a graphical window interface for user interaction. The window view presented to the project manager in the first state is shown in figure 5.12. Selection of table choices is made by pressing the middle mouse button when the cursor is positioned in the desired table cell. Mouse operations were standard for the different window views provided by the tools. Events only occurred in the window in which the mouse was positioned. The events were then selected when the middle mouse button was pressed. The process of highlighting various cells in a table can be continually repeated by clicking the middle mouse button when directly above a table cell. Only after confirming the operation in the bottom

section of the window is the last mouse click event taken to be the desired event. These operations were implemented by glueing graphic objects and statements outside the Objective-C language into class definition files.

Riskdriver 3

AVAILABILITY OF EXISTING PRODUCTS (OR PROTOTYPES)

WHICH CANACT AS EXAMPLES FOR DESIGNERS

The designers can refer to other products (or prototypes) having functionality which is -

identical to the required functionality	<input type="checkbox"/>
very similar to the required functionality	<input checked="" type="checkbox"/>
somewhat similar to the required functionality	<input type="checkbox"/>
no similar example is available	<input type="checkbox"/>
don't know	<input type="checkbox"/>

Quit

Figure 5.13: Amend Riskdriver Window

Default modes are associated with all windows, where user interaction can occur, therefore confirming an operation before selecting a menu option causes the default mode to be entered. Clicking on the abort table cell while the tool is in the first state will terminate current tool operations. Using the abort option in any of the other states causes the tool to go back one state.

5.5.2 Textual input

When viewing riskdrivers or Rules the project manager must specify the instance they wish to examine. By entering a numeric value in response to the operation mode prompt, the details of the associated instance are displayed. This is the only time while the project manager is using the tool, will input be accepted from the keyboard. Insertion of invalid numeric values or text will cause the appropriate error messages to be displayed.

Inserting information from the keyboard during phase one and two using the Objective-C code was performed with minimal fuss, using C scanf statements.

However the insertion of text using the graphic toolbox involved numerous statements to simulate a similar operation. Because the textual input was occurring inside windows created by the graphic tool box, special statements were required for entering text. Characters entered at the keyboard were recorded by the tool box as events of type 'Gevchr.t'. These events had to be repeated inside a loop until an escape character was entered. These events of type `Gevchr.t` then had to be converted into standard C character or integer types before being validated.

Writing code for something as simple as entering text, is clearly a contradiction of the Object Oriented principles discussed in chapter one. Entering textual details using the graphic toolbox proved complex and long winded. If complete Object Oriented systems are to be obtained, many improvements must be made to the interface package object. MacApp [Schb86], Appex[Cou 86] and EZWIN[Lib 85] are just some examples of Object Oriented graphic interface tools which could have contributed to a cleaner transition between the second and third phases.

5.6 Calendar View

Presentation of calendar information in graphic windows required the use of many objects used for displaying risk details. The rules relating to the physical layout of window used in the Risk tool, were used in the calendar tool. This meant that code for describing the bottom and top sections, as well as operations defining mouse operations, did not have to be written from scratch. Apart from changes to the string variables held by `icocar` the code from these sections is similar.

Displaying calendar details in the middle area required the introduction of a number of new techniques from the toolbox. The calendar, unlike the risk tool, cannot display all its selectable options at once. The calendar tool required a mechanism which allowed the scrolling backwards and forwards of selectable events. Allowing the user to click on specific events for a particular date, also meant that the selection mechanism should reference an event in two dimensions.

Using the lift objects which can be attached to a table, it was possible to have scrolling in both horizontal and vertical directions. The second problem was solved by setting up a two dimensional matrix (five by thirty one). The five rows representing the five events associated with a date instance, the thirty one is used to represent all date instances created by the Calendar tool. By attaching the matrix to a table object it was possible to have a table with items selectable in two dimensions. Attaching the table object to a box row object the desired

interface display was achieved. All five events could be displayed vertically in the window view, removing the need to have a vertical scrollbar. Restricting the table to only displaying five date instances however, justified the need for adding a horizontal lift (scrollbar) to the table.

Tool : CALENDAR		Project : IMPW			
	1989-2-28	1989-3-1	1989-3-2	1989-3-3	1989-3-4
Imports		***	***		
Products		***	***	***	
Meetings					
Personnel					
Work in progress	***	***	***	***	***
Quit					

Figure 5.14: Calendar Interface

Shaded boxes were originally chosen to represent events occurring on dates, however this facility could not be obtained using the current version of the graphic package hence the inclusion of asterisks. Clicking the middle mouse button inside cells with asterisks, cause the creation of another window displaying more detailed information(see figure 4.7). The creation of such boxes and the information obtained, are created using the techniques mentioned in this, and the previous two chapters.

5.7 Summary

This chapter related to the workbench interface required for the Risk and Calendar tools. Two major areas affecting the way interface operations were discussed. The automata described how a finite state structure could be used to show the various interfacing states of the tool. The automata machine provides a fail safe way for describing the various states associated with any mechanism. The proof of this reusability was the ability to use the automata for Risk and Calendar tools.

The graphic operations for displaying and interacting with tool windows were long and tedious. The code involved in the Risk class almost doubled due to the graphic package. Although the code describing the various interfaces was written and compiled with the other Objective-C classes, the graphic code written cannot be used for other applications, bar tools they were developed for. This type of development goes against the standard Object Oriented approach.

Chapter 6

Objective-C Traps and Pitfalls

6.1 Introduction

Developing computer systems in any programming language is never a trivial exercise. No computer language has been developed yet, which is “all things to all men”, solving problems quickly, using a simple and flexible syntax, whilst satisfying all performance issues. Objective-C despite all the benefits (inheritance, encapsulation and dynamic binding) is like other computer languages, in that it has shortcomings. The Object Oriented approach should be thought of as another tool to be added to the software designers toolkit. Like any tool in a craftsman tool box, its there for a specific purpose. Knowing when to use this tool and how to apply it to the task at hand, is a problem confronted by many computer professionals.

The performance factors of the Objective-C language in relation to conventional languages measures both negatively and positively, for various measurement areas. Factors such as space, speed, development time, software quality and code bulk are some of the issues which must be investigated before making a decision on the implementation language. This chapter will discuss how the Objective-C language used for developing IMPW tools measures against ordinary C for the above factors.

As with all computer languages it is important for code to be written in accordance with a certain syntax. The Objective-C language provides no exception, a number of compile and run-time faults were encountered during the development of the Risk and Calendar tools. The second half of this chapter will be

dedicated to faults which occurred while developing these tools, plus a mention of other factors which are important for correct Object Oriented development, such as inheritance. Knowledge of the foundation library's inherited classes is important for software quality, speeding up development time and reducing the amount of code written.

6.2 Objective-C Economics

Until now the Objective-C language has been talked about only in a positive sense. However Object Oriented Languages such as Objective-C like most things in life have a cost. The cost talked about in this chapter does not relate to monetary matters, but to computer resource costs (i.e. the cost on machine resources and machine efficiency). The amount of memory required by programs using conventional languages can in some circumstances be significantly smaller than Objective-C counterparts. There are numerous reasons for this differential in program size which we shall discuss.

6.2.1 Memory Costs

The first increase to program size that we will investigate is that caused by the actual object. Previous chapters have mentioned how objects inherit data and operations using the isa variable (which points to an object's shared part), to form the inheritance chain between the objects. All objects created into a computer system automatically inherit a isa pointer so that inherited details may be accessed. For small objects shown in figure 6.1.a this overhead is a substantial percentage increase on the memory requirements to that required for conventional languages.

Looking at this overhead for the object in figure 6.1.b the memory required for the isa pointer seems less significant. In large systems where thousands of objects exist, the extra space for objects in memory will also take up those extra thousand bytes of memory. This overhead of one byte per object seems trivial when we consider the amount of memory offered by commercial computers today, where four to eight mega bytes of main memory is common occurrence on many computers.

More serious concern about the space requirements are related to what the isa variable is pointing too, because all the details each object inherits will also

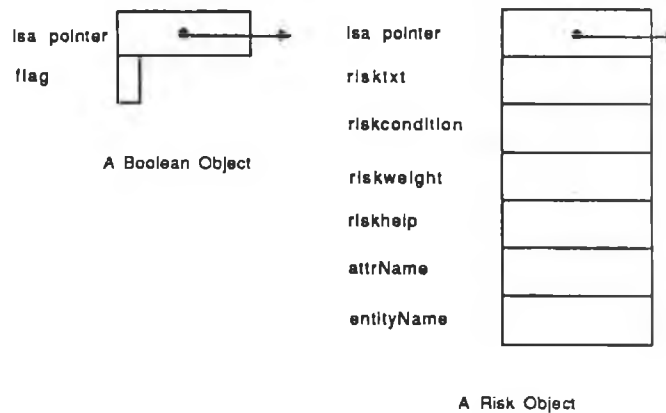


Figure 6.1: Object Overhead

be stored in computer memory when the application is running. The diagram in figure 6.2 highlights this problem by showing how the Objective-C objects must be linked together when a program is loaded into memory. Even if the only method required for an application is the `testMe` method in figure 6.2, all the other objects, plus the methods indicated on their dispatch tables must be loaded into memory.

For small applications this can lead to a substantial increase in program size, the root Object (inherited by all classes) alone takes up almost 40K bytes of memory. The greatest percentage of the functionality provided in such applications will usually be unused, because small applications would not require all the functionality provided by the inherited classes. Filer and float operation from the Object class alone account for 12K of memory, which in a lot of small applications may never be used. It is possible to delete code inside unused methods to reduce code size, however Objective-C users should make sure that the original library source has been backed up. Careful documentation of the different versions of Objective-C library classes must also be created to prevent other users from using foundation objects which do not contain the correct functionality.

When we look at larger Objective-C applications, the memory requirements of the Objective-C objects decrease greatly. This is because in larger applications where a great deal more functionality is required, greater amounts of code may be extracted from the inherited classes.

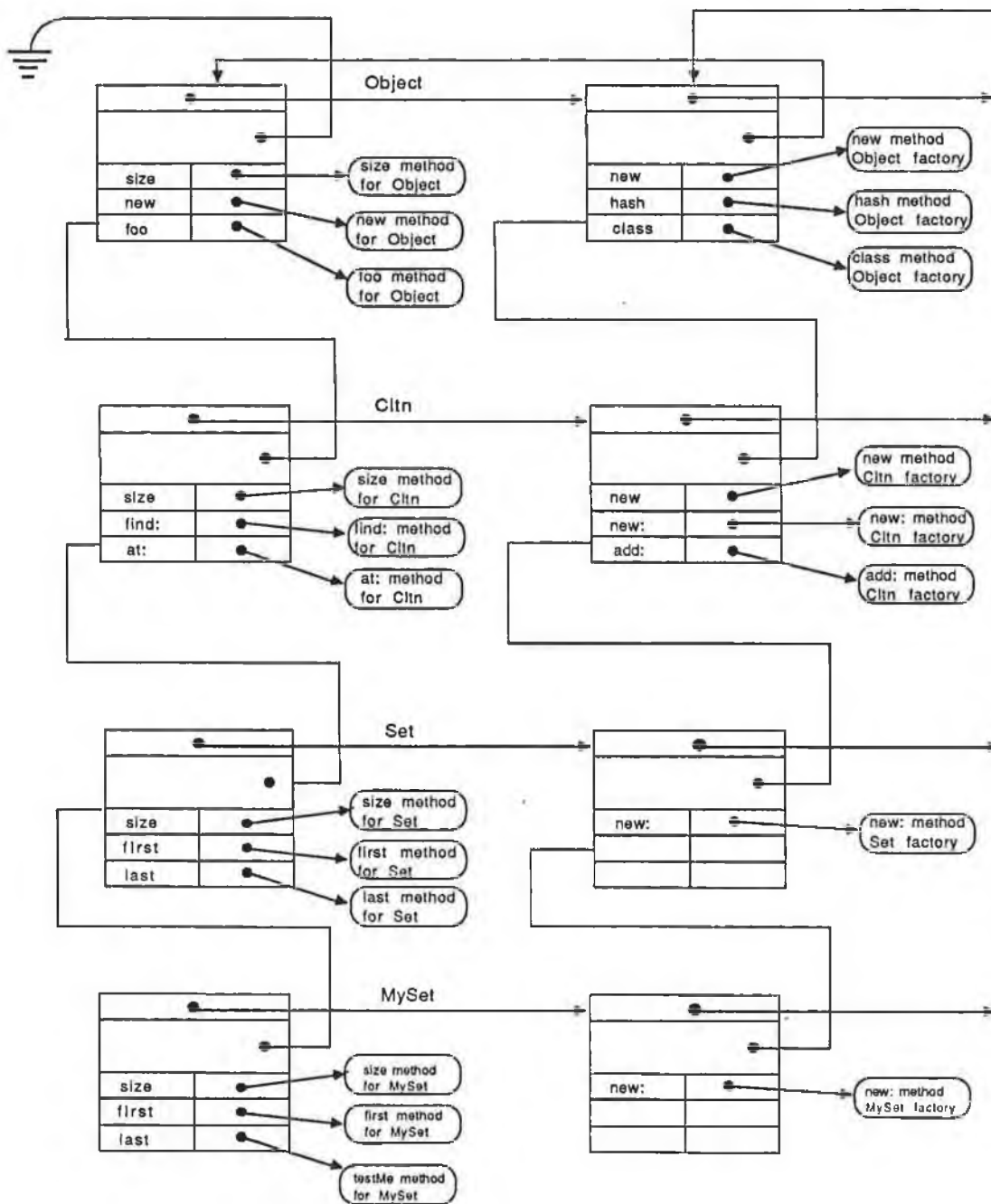


Figure 6.2: Objective-C required in memory

With conventional programming languages the size of programs grow linearly with functionality. This is because in most cases the code is being developed for specific applications and old code cannot be reused. The growth in size of Objective-C programs is relatively low in comparison, after paying the initial price of 40K, the extra functionality can be reused many times in larger applications. In large software projects Objective-C is often smaller than conventional code, this is through inheritance and the reusability of existing classes.

6.2.2 Code Size

To make comparisons between writing the Risk and Calendar tools in a conventional language such as C against Objective-C, would call for a substantial piece of additional programming. The development of such tools using standard C would require weeks of work, a luxury which is not feasible here, as would be the case with many medium sized computer projects.

In order to establish some type of measures I decided to convert a program which I had previously written in C into Objective-C. The program chosen was an AVL tree, which I developed while learning the C language. As with beginners using any new language, better ways of writing the code exist. It should be pointed out therefore that the results from these comparisons, are only used to give a rough approximation.

The time required to develop an AVL tree, which permitted only the addition of nodes to the tree was considerably shorter for Objective-C programs. This was due to the fact that all the AVL algorithms required were inherited from the SortCltn class provided in the Objective-C foundation library. The Objective-C program was developed in hours rather than weeks, as required for the C version. As an additional exercise I decided to develop an AVL tree which would allow the deletion of nodes. Writing the C code to perform this operation took the best part of a week, where the greatest percentage of time was spent debugging.

The Objective-C delete version took fifteen minutes, more time could have been spent on improving the layout and documentation. However this cannot overshadow the ease in which I was able to develop functionality for deleting nodes from an AVL tree (i.e. less than twenty five lines of code and less than fifteen minutes of work). Critics of Object Oriented technology may say that this example was tailor made for this type of problem. Other examples for small development applications made by StepStone the suppliers of Objective-C show similar trends. The results of these experiments are shown figure 6.3.

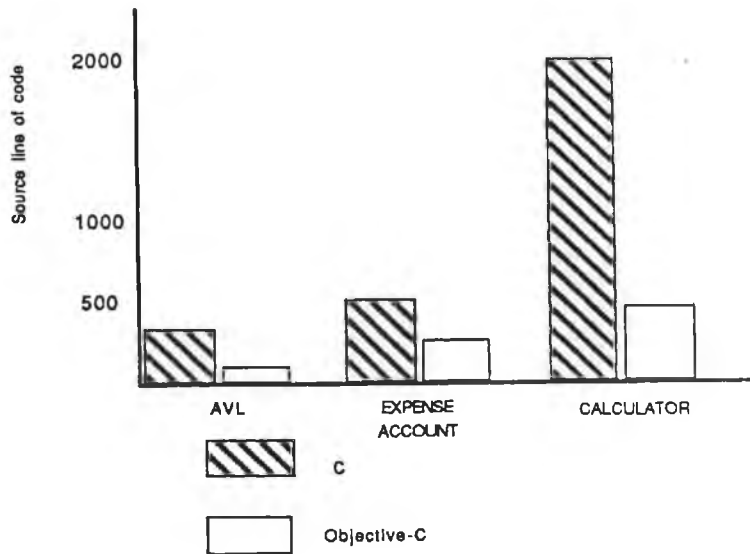


Figure 6.3: Objective-C productivity

The functionality provided not only reduces code size and development but also helps create applications where the standard of code quality is much higher than that of conventional languages. In the current example the AVL operations have already been written, tested and used in various computer systems. Therefore when we are using library classes there is no need to validate or document AVL operations. The Objective-C code is also more flexible, the changes made caused no change to the existing objects and code was reused from the inheritance network whenever possible.

The software quality provided helps to obtain a very high standard, while requiring only limited work (i.e. only code which is an extension of the foundation classes needs to be tested). Software quality using conventional languages such as C, are on the other hand dependent on the ability of the programmer and have to be re-assessed for each new application. No tangible means are possible to measure the development and testing time saved. However it is safe to assume that the Object Oriented approach provides enormous savings.

The standard C AVL program required 300 lines of code, a figure several times greater than that required by the Objective-C program. The code required for the delete extension for both programs showed similar results i.e. code for the Object Oriented approach was several times smaller than that of standard C.

6.2.3 Binary Size

The executable binary for both programs showed that the memory requirements of the Objective-C program were greater than that of its standard C counterpart. The main reason for this is because of the large amount of functionality added by the foundation library. As mentioned earlier, it is possible to reduce this execution bulk by replacing unused methods with dummy methods which perform no operations.

In larger applications such as graphic development, studies show that the differential between the two binary sizes are greatly reduced. The size of the Risk and Calendar tools ruled out the possibility of making the above comparisons. However the increase due to adding the graphic toolbox, had a more significant effect on the size of execution bulk in the IMPW, than the Objective-C code. For both tools the resultant binary size was increased by a factor greater than the order of two.

6.2.4 Messaging Overhead

An important factor surrounding the creditability of any language is the speed at which it performs its operations. The Objective-C language uses a messaging technique to access the required object functionality. Each class object contains this selection mechanism, where the class and selector name are hashed to give the implementation address. When an object sends a message, the object and selector are used in a hash algorithm to index the referred functionality. Message sending is slower than sending direct message calls, but faster than conventional condition statements. Figure 6.4 shows the differences in calling operations.

Obviously the messaging mechanism is going to be slower than direct calls to the functions as with the case statement above. Before the Objective-C code calls a function the method and selector values have to be hashed and accessed first. However this method still works faster than conditional statements, which may have to do tests on all possible choices before performing the required function.

Tests on the speed of direct function calls to C and Unix functions are rated between 2 to 2.5 times faster than doing the same processing using messaging. In order to verify these performance claims made about the Objective-C language, I created two small test programs to judge how exact these figures were. The first test was a simple print operation, the standard C program used `printf`. The Objective-C version used the `print` method in the `Object` class for printing the

```

Non Object Oriented case statement
CASE shape.tag OF
    circle    :    circleDraw geomFigure);
    square    :    squareDraw geomFigure);
    triangle  :    triangleDraw geomfigure);
ENDCASE;                                     // Fastest

Object Oriented
    [geomFigure draw];

Non Object Oriented conditional statement
IF geomFigure.tag = circle                // Slowest
THEN circleDraw(geomFigure)
    ELSEIF geomFigure.tag = square
    THEN squareDraw(geomFigure)
        ELSEIF geomFigure.tag = triangle
        THEN triangleDraw(geomFigure)

```

Figure 6.4: Message Overhead

same data which was created by the String class. The second test was converting an ASCII string into an integer. Ordinary C used the `atoi` function, while the Objective-C language used the `asInt` method which was attached to the string class. Because of the size of both tests a large number of iterations had to be performed to establish timings. The results of these tests are given in figure 6.5, all results given were returned using the Unix `time` facility.

The real figure represents the total elapsed time for running each test. The user time represents the time spent executing the test program and sys time is the time spent executing in the operating system during system calls. Testing the printing of a string showed that the Objective-C performed better than the approximation given. However the conversion of ASCII to integer values provided values closer to the approximations given above. The fact that Objective-C is a hybrid language has the advantage however of allowing direct C function calls to be made in certain circumstances. As a result, the overhead for implementation is often much less than the figures mentioned above. Reductions in the messaging

function	iterations	ordinary C			Objective-C		
		real	user	sys	real	user	sys
print	1000	5.6	0.2	1.0	7.3	0.2	1.2
	10000	43.7	2.1	9.0	47.6	3.1	10.1
atoi	1000	0.3	0.1	0.0	0.8	0.5	0.0
	10000	3.6	3.5	0.0	6.7	6.5	0.0

Figure 6.5: Objective-C performance

overhead can also be realised by reducing unnecessary message calls. The example in figure 6.6 shows two different way of writing the same method.

The method "list" in figure 6.6.a, performs a message call for each execution of the while statement despite the fact that `aRiskCltn` value will be constant throughout the looping operations. The correct solution shown in figure 6.6.b, performs only the necessary message operations. If `aRiskCltn` contains hundreds of risk objects, then hundreds of message calls would be saved by only asking for `aRiskCltn` size once.

6.3 Error Clinic

The Objective-C language like C is terse, expressive and is designed to be easily used by experts. There are few restrictions to keep the user from blundering. The remainder of this chapter will talk about errors I have made while using the Objective-C language, plus some which I have avoided, but which all Objective-C programmers should be aware of.

Objective-C being a hybrid of the C language may contain any of the errors associated with standard C programs, plus those associated with the new type `id` and the new operation the message. Errors associated with ordinary C are very much a part of the problems encountered while developing Objective-C programs. However this text describes errors in relation to Objective-C. Readers interested in more information on errors related to the C language, may find [Koe 86] interesting reading.

```
(a)  - list {
        int i = 0;
        while (i < [aRiskCltn size])
            [[aRiskCltn at:i++] print];
        return self;
    }

(b)  - list {
        int i = 0;
        int risksize = [aRiskCltn size];
        while (i < risksize)
            [[aRiskCltn at:i++] print];
        return self;
    }
```

Figure 6.6: Saving messaging time

6.3.1 Class Definition Troubles

One of the early problems for Objective-C beginners is getting their programs to compile. The Objective-C syntax for each class definition file requires writing code in accordance with a number of predefined conditions. Before each class name definition the “=” symbol must appear, while after the class name a colon must be entered before the parent class is written. Failure to enter any of these symbols will cause compilation errors.

In each class definition file, the message groups associated with the class plus other classes referenced in this file must be declared. The message groups are required to keep track of the methods return types. StepStone supplied three message groups Primitive, Collection and Geometry to watch over the return types in the methods supplied by their foundation library. Classes created by the user must set up their own message groups to keep track of the types returned by the methods in these classes.

Mistakes such as leaving out a message group or using the wrong message groups are problems common to many Objective-C programmers. For example it

is wrong to include only the Collection message group when creating a new class which has `OrdCltn` as its parent. Not including the Primitive group will cause errors to occur at runtime. Because the Primitive group keeps track of all return types used by the root class `Object` which is inherited by all the other classes, therefore this group should appear in every message group declaration.

Another frustrating error which can occur when compiling class definition files is forgetting to print the `"=:"` at the end of the file. This symbol will have no association with the logic of the class file but its absence at the end of the file will cause error message which can add to confusion for the naive Objective-C user.

6.3.2 Erroneous Methods

Defining methods inside a class definition file can cause errors which are either syntactic or semantic. The errors due to syntax are usually the simplest to correct. The compiler will list the line number and give a diagnostic message indicating the type of error that has occurred. This could be for a misspelt variable, a missing semi colon at the end of a line, no matching braces or brackets inside statements.

The methods defined inside class definition files should always commence with a plus("+") or minus("-") sign to represent the factory and instance objects respectfully. If no type is declared after the method sign the default is assumed to be of type `id`. People more accustomed to developing in the standard C environment expecting an integer value to be returned from an undefined method definition, may encounter runtime errors if they are hoping to perform some operation on the returned value. The Objective-C language also requires that the left curly bracket which signals the start of a procedure should appear on the same line as the method definition name. It is an error to create methods using syntax similar to figure 6.4.a, the correct definition is shown in figure 6.4.b. The type returned by this method is the default `id` (object identifier).

The `self` and `super` pseudo variables are important for identifying objects and overriding methods in classes. Trouble due to incorrect use of these variables has already been mentioned. Using `self` instead of `super` as an object receiver for a selector with a similar method name will cause the method concerned to enter an infinite loop as shown in figure 6.4.a.

The errors associated with methods which really stilled me up were those

(a)	(b)
<pre>- test { // int i; [self test]; " " " " return self }</pre>	<pre>- test { int i; // [super test]; " " " " return self; }</pre>

Figure 6.7: Method Syntax

that occurred at run time. Returning and passing incorrect types was one of my most common errors while learning Objective-C. Any methods or variables which had no type declared were given the default type `id` and not integer as in C.

6.3.3 Main Module Structure

Objective-C uses a main program for the initiation of code like conventional C. Indeed the overall structure of both is similar. The only differences would be the inclusion of Objective-C statements if any, used inside the main module, plus the class and message group declarations. The main Objective-C module declares the message group used by the file, as is required in any Objective-C file that sends messages.

After the closing bracket in the main program two additional lines with Objective-C syntax are added. The command `@classes(class list)` is used to declare the classes used by the file. The command `@message(message list)` declares all the messages used by the classes. If the `@classes` statement is in the same file as the message statement, the list of messages(which can be quite large) can be excluded.

The compilation of class files must occur in a generic order. You cannot compile a class if the parent class which you have also created has not yet been compiled. The main program module which initiates the various classes must be

compiled last, before the linking of binary objects. The addition or deletion of methods from a class should be followed by the re-compilation of the class file and the main module so that the message table containing all the returned types is always up to date. Concentrating on other system problems can often lead to this simple procedure being overlooked. The Unix “make” command provides a foolproof way around this problem. It is possible to set up a make file which will always re-compile the main module when changes have been made to the value returned by a method.

The Objective-C language uses a number of include files for various implementations. The “objc.h” file is the standard header file used by the language, containing the most common definition types and macros used by the language. Failure to reference the objc.h file at the beginning of a class definition file using the Objective-C type such as BOOL(Boolean) will result in a compilation error. The naive user will find the diagnostic message associated with this error difficult to comprehend. The syntax for the error will vary in accordance with the first line of code which required the header file. The line number associated with the error will usually be one greater than the number of code lines in the class definition file.

6.3.4 Printing Errors and Error Messages

If you wish to print all the instance variables associated with an object a method specific to printing this type of object is required. Trying to print an objects data using statements as below will result in compiler errors.

```
printf("%s\n",self);
```

When wishing to print an individual instance variable associated with an object, statements of the form

```
printf("%d\n",[self testSize])
```

should be used. However this may not be sufficient to prevent the program from crashing, if a method has not been written to return the correct type. When creating new classes it is good practice to enter immediately a method with its correct return type for each instance variable created in the class.

During the development of both IMPW tools, runtime crashes often occurred because objects were sending messages to methods which did not exist. The compiler does not perform any checks to decide which methods belong to which object. It is the programmers responsibility to ensure that these dynamic objects only send messages to methods that will understand them. The diagnostics associated with sending invalid messages, will give information similar to below when this error occurs.

```
2a2a4=Fruit[Fruit@0x2a2a4 error:]Does not recognise selector colour
=== stack backtrace (in reverse chronological order) ===
=== [receiver selector args] @sentFrom[ClassName methodName] ===
    <function(2a2a4,209f8)> @2d96[Object -error:]
    [2a2a4=Fruit -error:209f8] @2d70[Object -doesNotRecognise:]
    [2a2a4=Fruit -doesNotRecognise:2008f] @342e(non-method)
    [2a2a4=Fruit -colour] @2486[Fruit -printOn]
    [2a2a4=Fruit -printOn:217c8] @2786[Object print]
    [2a2a4=Fruit -print] @20d6(non-method)
```

The error diagnostics gives a trace back of all the methods called in relation to the invalid message. In this example, the “colour” message sent by the Fruit class was not found.

Applying messages to arguments caused additional problems while learning the language. The colon used by selectors for indicating arguments can become confusing for methods containing a large number of arguments. Passing incorrect arguments types and setting up the wrong method definitions were also problems encountered while using the language.

6.3.5 Collection Errors

Both the Risk and Calendar tools used the OrdCltn class extensively. One of, the most common errors to occur when using this class was the out of bounds error(i.e. trying to access an invalid offset in the OrdCltn in question). Another problem was accessing objects which the OrdCltn did not recognise. By adopting a good programming style it is possible to avoid these problems. Sequencing over a collection in the following manner always guarantees printing valid objects(provided the objects know how to print themselves).

```
id riskCltn = [OrdCltn new];

riskSeq = [riskCltn eachElement];
while(aRisk = [riskSeq next])
    [aRisk print];
```

Looping through a collection using this technique makes it impossible to go out of bounds. Relative accessing of objects using the `OrdCltn` statement:

```
[riskCltn before:bRisk];
```

generates an error if `bRisk` cannot be found. Accessing an object using the `at:` method is an alternative technique for retrieving or placing objects, but when using this approach the user needs to ensure that the integer value given is inside the collection range. This type of error can only be eliminated by validating the `OrdCltn` argument before it is applied.

6.4 Garbage Collection

One of the main problems related to the Objective-C language was garbage collection. The 3.3 version of the language used for developing the IMPW did not provide garbage collection. Unlike languages such as Smalltalk-80, the software developer is responsible for the deletion of objects when they are no longer required by the system being implemented. This responsibility puts an extra burden on the person developing a system. The absence of such a facility means that complete knowledge of the application environment is required so that objects maybe safely deleted. In large applications where a great amount of work is inherited or performed by another person the work involve is substantially increased. Deletion of objects, containing other objects, such as `OrdCltns` must have their contents deleted before the collection object is deleted, otherwise the system will contain a number of objects dangling in memory, which may cause space problems or side effects caused by addressing the memory occupied by one of these objects.

6.5 Inheritance

The inheritance mechanism introduced many benefits with Object Oriented languages, throughout this thesis many references are made underlining these benefits. However the inheritance mechanism which is used to provide reusable and high quality software also introduces some additional problems for the naive user. The Objective-C foundation library contains twenty-eight classes and approximately two thousand methods. Before commencing any development work, a general understanding of the class hierarchy and the operations available is required.

For someone using the Objective-C language for a specific application, knowing about all the available classes and their operations may seem a fruitless operation, when probably they only need to know about two or three of the classes. But if the user is serious about using reusable code, some type of understanding of the available class inheritance is required, to avoid rewriting unnecessary code.

For Objective-C beginners probably the best way to start programming is to learn about the root class `Object`, the overall inheritance structure, and a quick overview of the functionality provided by the other classes. Only by reading about the classes and using them in software development will the user get a real feel for the class library. The compiler suppliers, also supply additional classes in packages called "software-ICs". Users looking to optimise reusability in their system may purchase any of these packages if relevant to their system development.

6.6 Summary

The syntax and semantic errors discussed here are only the tip of the iceberg. As with any computer language the only way of understanding the syntax is through practical experience. With this experience the compile time error should become nothing more than a temporary annoyance. The real problems will be those that execute correctly nine times out of ten or crash every time you are showing someone a demonstration. Problems related to inheritance (i.e. what is available and which class to inherit from) will also be improved with reading and practical experience.

The problem of garbage collection is one of the main black spots of this

language compared to others such as Smalltalk-80 and Eiffel. Future versions of the Objective-C compiler have been promised to handle this problem, however for users with versions up to 3.3 of the compiler, the problem of handling an objects life span remains their responsibility.

Chapter 7

Object Oriented Design

7.1 Introduction

Before commencing programming with any computer system, a methodology for the design of the new system is needed. There are numerous ways of approaching the analysis of systems. Many of these methodologies were developed during the late 70's and early 80's, reflecting an upsurge of activity in IS development. However most of these methodologies were designed for system implementation using a third generation computer language, characterised by a linear approach to systems analysis.

Because the conventional methodologies provide no mechanism to support Object Oriented characteristics such as encapsulation and inheritance, new modelling techniques for Object Oriented languages were developed. As with conventional system modelling, there have been a number of approaches put forward to help system building using Object Oriented languages.

Because the Object Oriented technology is still in its infancy(at least in commercial terms) no standard approach to Object Oriented Design has yet been agreed upon. Some of the new approaches towards Object Oriented Design seem geared to the creation of software systems using particular languages, which do not illustrate the full power of most Object Oriented systems. Other approaches try to merge the conventional techniques along with new Object Oriented techniques. This helps designers create systems in a manner which is similar to the already proven techniques while using the Object Oriented mechanisms, however such approaches represent language inheritance poorly. This chapter will inves-

tigate some of these techniques, showing how they could be used in modelling software systems similar to the IMPW tools.

The variety of applications in which Object Oriented techniques can be applied to cover all problem domains from developing a process control system in a nuclear factory to creating a system used for counting apples in a basket. The Objective-C suppliers provide reusable and extendible components for implementation. But they do not provide any assistance towards the development of other reusable classes.

The techniques for designing reusable and extendible components have not advanced at the same rate as the Object Oriented technology. This topic has become a popular research area for many Object Oriented enthusiasts, but as yet no global accepted model has been accepted for Object Oriented Design. This chapter looks at some of the design models which have been developed to solve this problem, and the approach taken for the development of the IMPW tools.

7.2 Booch Model

One of the earliest approaches to tackle Object Oriented design was the Booch model[Boo 86], named after its inventor Grady Booch. This approach decomposes the textual description of the system requirements to develop the class objects for the problem domain. Nouns in the textual description of the system, relate to object classes which have to be created. Verbs identify the operations performed by the objects. This process can be continually repeated to decompose objects to lower levels of abstraction. The approach recognises operations associated with objects and the operations they require from other objects.

The foundation of the approach is based upon information hiding(encapsulation) and data abstraction. The complexity associated with large systems is removed by representing problem domains as abstract objects, which may communicate with other objects in a manner which cannot be viewed as sequential.

The diagrams used to represent these objects and their inter connections are known as Booch-grams, which are sometimes compared with conventional data flow diagrams[Woo 82]. The sources or stores of the data flow diagrams can be directly mirrored to objects in the Booch-gram. Further investigations of the data flow processes leads to the identification of more detailed processes which occur at lower levels of abstraction. A similar approach is used to determine objects in Booch-grams, where the investigation of object detail may identify the

need for lower level objects. After understanding the requirement associated with a system the Booch method is applied using the following steps.

- [1] Identify objects and their attributes (All nouns in the textual description of the requirements are used to represent objects, the verbs represent the actions on these objects.)
- [2] Identify Operations (Allows objects to be decoupled)
- [3] Establish Visibility of each object in relation to other objects (This helps to provide a generic structure to the objects in the system, capturing the topology of objects in the model)
- [4] Establish interface of each object (Establish the specifications to be performed by the module and the views that will be displayed to external objects)
- [5] Implementing each object (Choose a suitable representation for an object, in most cases this will involve decomposition of the object until the operational level is reached)

This approach is more responsive to change than traditional methods because the changes to objects are more localised. Different levels of abstractions can also be obtained for each object by repeating the process which obtained the top level view. Using the Booch-gram approach in the Risk Analysis tool gives a top level model as shown in figure 7.1.

The model also provides a better indication of the flow of control than traditional approaches, where there is one single thread of control which is operated in a sequential manner. The Booch-gram model gives a better representation of the nature of concurrency attached to a system. Perhaps one of the most important benefits from this approach is the mechanism it gives to formalise our model of reality.

Unfortunately most of the Booch-gram displayed in computer journals are usually directed to the development of systems written in Ada, an Object Oriented languages, which does not support inheritance. For languages such as

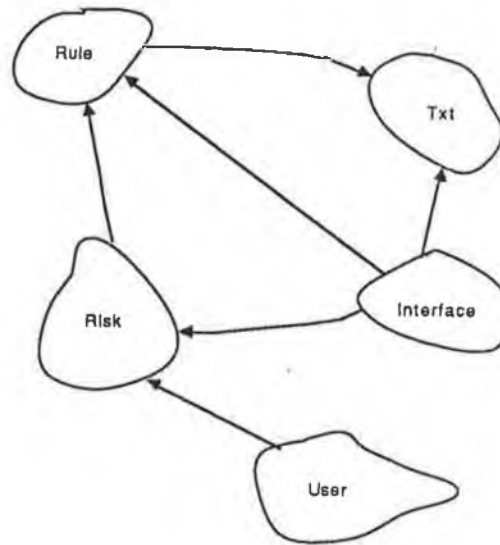


Figure 7.1: Risk Booch-gram

Objective-C, C++, Smalltalk-80 etc, this approach makes no attempt to make use of the power of inheritance. The approach ignores the numerous high quality abstract classes and operations supplied by these languages and hence fails to give a fair representation in the model view.

7.3 Hierarchical Object Oriented design(HOOD)

Another prominent approach is the Hierarchical Object Oriented design method called HOOD[Rob 89]. This approach like Booch's, is aimed for software development in the Ada community. The structural design techniques used in conventional methodologies is highly utilised by this approach, allowing the designer to use techniques that are already proven and tested. Like Booch this approach is centered around identifying the objects which will map the real world objects into software entities. The HOOD design can be broken down into four phases.

- [1] Definition and Analysis of the problem
- [2] Revise into the design solution i.e. natural English (This is written in a semi-formal style so that the objects can be selected for the next phase)

- [3] Selecting the objects and operations (Similar to Booch i.e. nouns are taken to represent objects and verbs relate to operations on the object)
- [4] Refine the design to produce a more formal description of the object interface. (At this point all the variables are related to an object)

This approach recognises two different types of objects, passive and active, which are required for modelling the software system. Passive objects are those executed immediately when control is passed to the object, while Active objects execution is dependent on the object control structure. Diagrams used for representing these objects are shown in figure 7.2.a and figure 7.2.b, Active objects are denoted with an A in the top left hand corner.

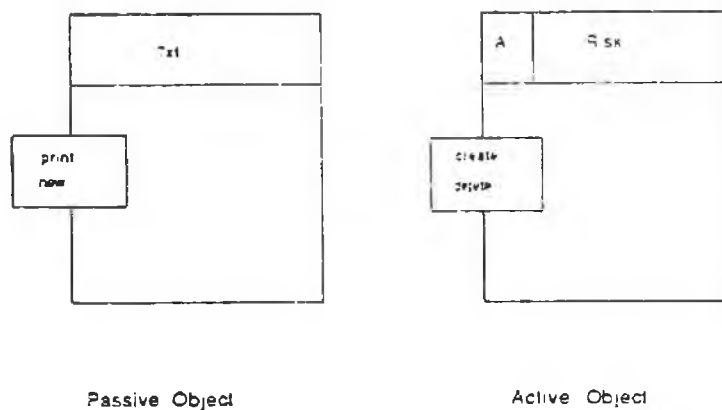


Figure 7.2: HOOD diagrams

The object names are displayed along the top of the boxes and the operations which may be performed by the object are inserted in the box which is placed on the perimeter of the object box. For an object such as Rule in the Risk Analysis tool, where a large number of operations are performed, this box proves an insufficient way of representing this information. Another problem with this notation is that it makes no attempt to identify the parameter types passed or returned from these operations.

With many Object Oriented languages it is feasible for objects to call operations defined in other objects and to inherit data plus operations from parent objects. The HOOD model recognises these features and has built techniques into the approach to handle such cases. Objects using operations inside other objects can be represented by the USE arrow as shown in figure 7.3. However the approach does not permit cyclic calls between passive objects. Therefore if the Rule object uses one Risk object operation, the reverse operation would not be permitted if both were passive objects. Another problem with this facility is that although it indicates that an object requires an operation from an object, it will not determine which operations are required, when more than one operation is available.

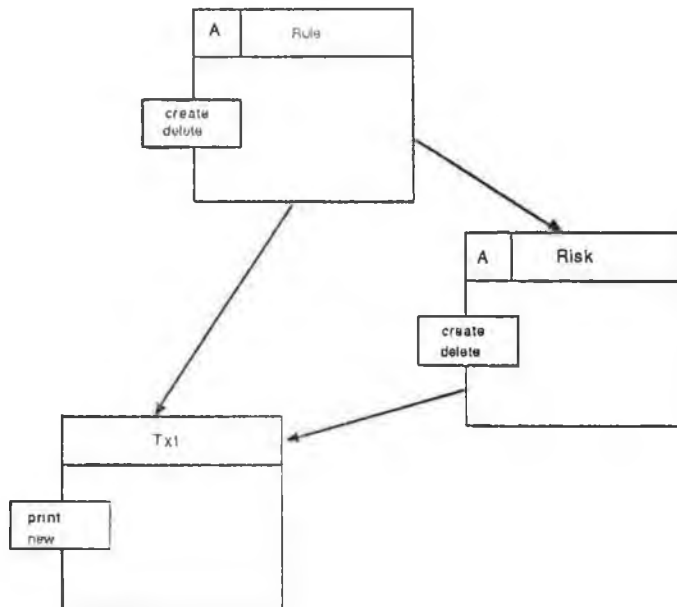


Figure 7.3: Message passing with HOOD

Inheritance is represented by showing the child object(s) inside the parent objects see figure 7.4. This provides a useful technique for representing objects at the same level of abstraction. Each level can be repeatedly decomposed until the abstraction process is exhausted. Once the diagrams representing the software system have been completed, many HOOD tools produce code translating these diagrams into Ada code. This approach like the first, sets up a foundation for a design which can be easily extended. However like the Booch approach it also ignores the class hierarchy supplied by many Object Oriented languages and does

not suggest any way in which the design can be reused.

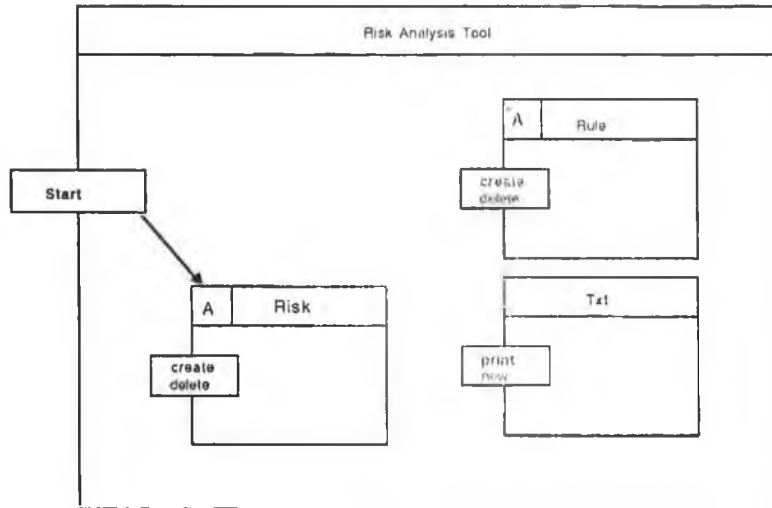


Figure 7.4: HOOD inheritance

7.4 Block Design

The Block design approach was developed for the creation of large projects [Jac 83] using Object Oriented technology. The block design method merges Object Oriented techniques and conceptual modelling used for requirement modelling of information systems. The approach unlike those previously mentioned, focuses on the interconnection of reusable software components. This concept can be considered similar to Objective-C software-ICs, where instead of designing from scratch, the design components can be looked up in a catalogue and entered into the design.

The blocks provide the framework on which this approach is based, each block representing a package service of the system. Blocks decomposed to their

lowest level, represent classes in the Object Oriented languages. The components described by the approach are used to represent standard modules that can have many different applications. These components are available to all the blocks used in describing a software system. This facility can be considered analogous to the foundation library offered by some Object Oriented Languages.

The concept of inserting a specific design block to meet certain conditions agrees with the reusability notion put forward by Object Oriented technology. The approach involves a number of steps from the original requirements, to the finished system. Progression is made from system analysis to design, using a number of facilities similar to those mentioned in the HOOD approach. However the transitions involve moving through the various steps which can lead to a model which is not error free. The approach is geared more towards developing large systems, where the duration of the project is ten man years or more. The design is broken down into numerous parts to allow the participation of numerous designers. While cohering to many of the Object Oriented principles this methodology can seem cumbersome when designing small to medium size applications.

7.5 Object Oriented Structured Design

This approach is a combination of traditional approaches to which Object Oriented concepts have been added[Was 88]. Developers using this method for system design find themselves using the "top-down" approach when performing functional decomposition of modules. The "bottom-up" strategy is more likely to be applied to objects, where developers will be more concerned about the functionality of the object than its overall role in the system design.

The structured approach puts a strong emphasis on modularity, so that the system being developed will be comprehensible and flexible, highlighted by the fact that modules can be created and tested independently of each other. Object oriented design is merged into the approach to represent the hierarchy of objects. By keeping modularity the principal component of design, development is made on already proven concepts. This approach is also useful in allowing the method to be used in a number of various applications.

For designers using either Booch or HOOD methods this approach provides a great deal of familiarity, hence avoiding the need to develop the necessary design skills from scratch. The basic notation used in structured design are used to illustrate modules. Applying these structured design concepts to the Calendar

tool, provides the layout shown in figure 7.5.

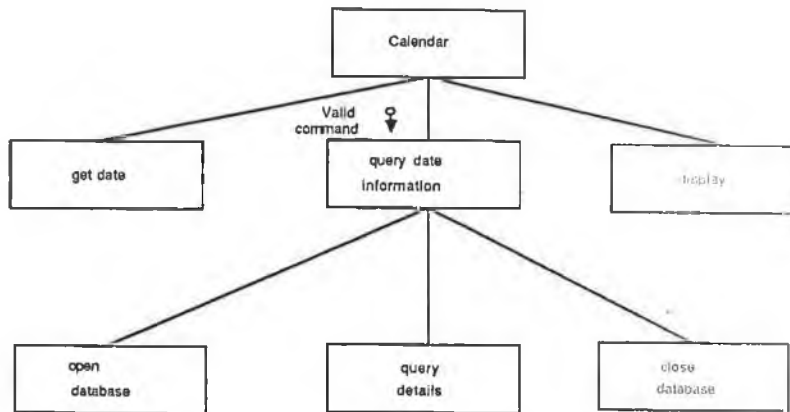


Figure 7.5: Object Oriented Structured Design of the Calendar

The sequencing of operations in this diagram are taken from left to right, with the operation at each level giving more specific detail than the module at the higher level.

The concept of object and the operations they perform is described in a manner similar to HOOD. The object is characterised by a box, with the associated operations appearing in smaller boxes around the perimeter of the object box. The encapsulation of data and operations by the approach is termed "information clustering". Unlike the previous approaches, a distinction is made between the definition and uses of objects. The definition of an object is represented by a rounded rectangle which is displayed inside the object box as shown for the Risk object in figure 7.6. The operations used by the object are represented by individual boxes which are positioned on the objects perimeter, making it possible to show the parameters associated with each operation.

Inheritance and Encapsulation, the two facilities desirable in most Object Oriented languages are represented by this approach. The inherited object in figure 7.6 is denoted by a dashed box, the inheritance of operations are also denoted by the dashed line between the boxes. The Risk object in this diagram overrides the add and delete operation inherited from the object class.

Another nice feature about this approach is the way it allows a programmer to gradually shift from functional decomposition design, towards an approach which encompasses Object Oriented concepts. Analysis of the problem domain may be made by a number of various methods, before being translated into the design model. Unfortunately describing Object Oriented systems with functional designs can cause problems when making extensions. Updating conventional functional systems often lead to changes which are not restricted to localised modules. While facilities for describing inheritance have been provided, the approach offers no advice on how this structure is set up. It presumes that all designers should know intuitively about the languages foundation hierarchy structure.

7.6 Learn by Example

Besides learning to design Object Oriented systems using one of the aforementioned methodologies it is also possible to learn design techniques by copying previous examples i.e. Learn by example. By studying the structure of previous Object Oriented systems and following the inheritance path for data variables and operations a person can develop a better feel for design. This approach can prove useful for learning about the foundation classes provided with a language and helps to optimise use of the inheritance mechanism. This technique provides no standard way of describing the model other than coded listings, unless the language provides a tree or browser mechanism as in Smalltalk-80. It would probably be better if used in accompaniment with another design approach rather than by itself. Another problem is that studying old systems can also be very dangerous, if the software being examined has not been designed properly.

7.7 Methodology for workbench tools

With no proven standard for Object Oriented design, the technique used for the development of the IMPW tools was not related to any one individual methodology. The initial design used IDEF-SADT approach, to describe the IMPW layout, however this technique could not be used for creating reusable Object Oriented tools. The approach taken was however influenced to a certain degree by a combination of the various approaches available.

The creation of both Object Oriented tools for the workbench followed the

following phases.

- | | <u>INHERITANCE</u> |
|-----|--|
| [1] | Initial Requirements and Analysis |
| [2] | Develop Objects to model real world requirements |
| [3] | Find the operations related to each object |
| [4] | Convert operations to objects if feasible |
| [5] | Repeat steps three and four until all operations are associated with a class |

The first step in the creation of the tools, involved the analysis of the system requirements. This section concentrated on how the requirements for the Risk tool were fulfilled using this approach(similar operations were used for the creation of the Calendar tool).

After completing the analysis of system requirements, the objects which matched the real world entities were created. The techniques used for developing these top level Risk tool objects were analogous to that mentioned by Booch i.e. the requirements are read in natural English, with the nouns representing class objects and verbs the operations. The diagram in figure 7.1 shows the top level objects which were created for the Risk tool.

To define all the necessary top level objects after studying the initial requirements puts too great an emphasis on this phase of modelling (i.e. the need for certain objects may only be highlighted by viewing the operations of other objects, or they may be simply overlooked at the start). Only by iterating through the various phases, is it possible to ensure that all the necessary objects are defined. This approach may be viewed as an Object Oriented prototyping exercise, where each iteration brings the user closer to the desired model.

The third step in the process identifies the operations required for the object. Abstract or reusable operations found at this stage suggest the creation of other classes. Detailed study of these operations by the user, may lead to the

identification of new classes undetected by previous studies of operations and objects. The technique used for finding the functionality associated with the Risk tool objects, used the Jackson structured method, where each operation was identified as a module. Each abstract object of the Risk tool was modularised in a manner similar to the Risk class in figure 7.8.

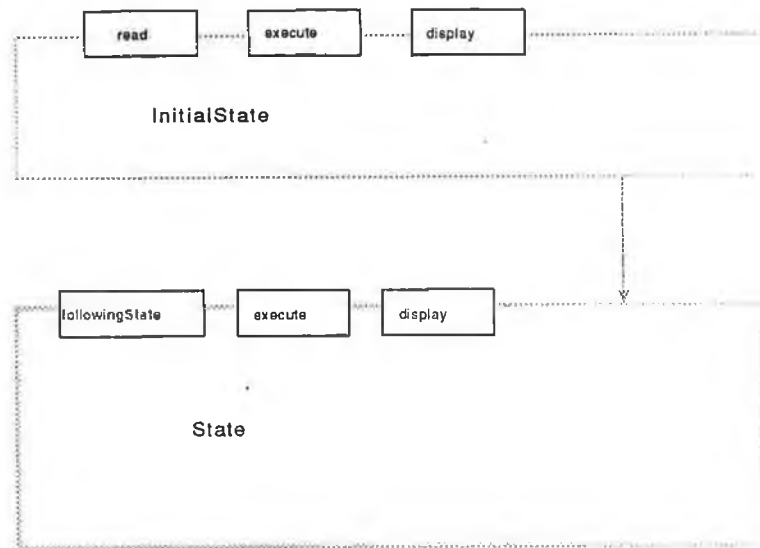


Figure 7.6: Object Oriented Structured Design for inheritance

The technique used for describing the functionality should be dependent on the system developer (i.e. choose the methodology you feel the most comfortable using for this task). The reasons for choosing the Jackson Structured methods [Jac 83] here, were twofold. First it seemed the simplest and most natural way of describing the functionality of the abstract object. Secondly, the method helped to differentiate between Objective-C factory and instance methods. All the top level modules in the Jackson diagram describing an object can be identified as factory methods i.e. class operations. Operations described by lower level decomposition of the modules represent instance methods. Therefore the operations add, delete, list etc, are considered factory methods for the Risk class.

By iterating through these operations for each object, the system designer is able to develop an intuitive understanding for the relationship between objects and operations. This is important not only for comprehending how objects will communicate, but also for identifying areas where code can be reused.

The power of inheritance has been poorly mentioned in this approach, as

is the case with many of the methodologies mentioned so far. However this is a prominent factor when developing systems using Object Oriented Languages. The inheritance tree must be known during design, in a way mechanics know every tool in their toolbox. The mechanic is aware of the tools available in the toolbox, knowing exactly when and where they should be applied to the task at hand. Similar knowledge of the classes and operations available, plus intuitive knowledge to where they should be applied during system design are deemed important for successful designing.

However this method is a contradiction to the principal that design should be independent of the language, because the technology has no established standards as to what objects should be contained in an Object Oriented library and how it should be structured. At present all the software suppliers want to insert classes into libraries, instead of reusing classes already there.

This approach while being adequate for the isolated design of the workbench tools, falls well short in answering all the Object Oriented requirements. The Object Oriented design methodology applied to the system, should remain language independent, while also being reusable and extendible. By iterating over the objects involved in system creation, the validity of the classes being reused in other systems can be questioned (if the iteration process occurs on objects used in different applications, the word re-worked becomes more appropriate than reused). The classes developed for the Risk and Calendar tools did not render any reusable classes. Perhaps using another approach or better design of the classes themselves, this reusability factor could have been obtained. The next section will concentrate on the reusability of classes issue.

7.8 Designing Resusable Classes

In addition to the problems of knowing what classes to inherit from, where to attach new classes in the inheritance hierarchy, it is important to design classes which conform to the Object Oriented principles of reusability and extendibility. The techniques used for the development of the Risk and Calendar tools were designed purely to satisfy specific implementation on the IMPW, using the class facilities offered by the Objective-C foundation library. As Drake[Dra 88] observed, the Object Oriented technology will separate programmers into two groups, those developing abstract classes, and those developing system applications from these class libraries.

The objects created for the Risk and Calendar tools cannot claim to be

reusable in the sense that they can be added to the foundation library and used in various applications. These classes referred to as modular units by Meyer[Mey 87] have to satisfy a number of criteria if we wish them to be reusable and extendible.

- **Decomposability** It must be possible to decompose a complex module into smaller sub modules.
- **Composability** Modules should have the ability to be applied as independent elements in the construction of other systems.
- **Understandability** If a module cannot be understood independent of other modules, other modules will be involved in the maintenance of this module.
- **Continuity** Changes made in a module should have little to no effect on other modules in the system.
- **Protection** Isolation of errors within a module thus stopping the propagation of these errors throughout the system.

It is the composability factor which prevents the factory objects in the workbench from been reusable. The variables and operations described relate to a precise application.

However when creating application specific software, designers should still be trying to develop classes that can be reused in other applications. By adopting a number of rules of thumb it is possible to create classes which may be used in different application environments.

- [1] Avoid declaring unnecessary instance variables into abstract objects. The declaration of these variable can make the class object too specific, at too high a level.
- [2] Use the same message protocol for describing operations as described by other classes in the foundation library.

- [3] When deciding which superclass a class should inherit from, method inheritance should be given preference over data inheritance.
- [4] For classes which have half their variables accessed by one half of its methods and the other instance variables by the other half of its methods, split the class into two, each with the required instance variables and methods. Attach both classes to a higher level abstract class.
- [5] Code defensively! If you want your code to be reusable be prepared to have your methods called under various circumstances.
- [6] Verify that your testing and documentation can be used in various applications.

Some of these rules are simply common sense, others may only be found and appreciated by getting one's fingers burned creating factory objects. The main reason for the IMPW tools not producing reusable classes was mainly due to a lack of development experience and geographic distribution of tool builders rather than the specific applications for which the tools were being developed. As developers create new classes using Object Oriented technology, the initial code size will grow linearly with the classes being created. However as programmers gain experience, the size of the code in relation to classes should gradually decrease as the programmers bring reusability into play.

7.9 Summary

This chapter has viewed a number of Object Oriented Design methodologies, none of which satisfy all the criteria required to describe system building using Object Oriented languages. Many software suppliers are steamrolling ahead advertising their Object Oriented language and the class facilities provided, ignoring the problems of design and standards. Techniques such as HOOD and OOSD are a positive step towards a correct Object Oriented Design approach. However work is still required to make these design components reusable in different applications and languages.

Chapter 8

Future Directions

8.1 Introduction

During the past two decades computer professionals have seen the invention of many tools used to combat problems that plagued the computer industry. Software libraries tailored for specific applications, more powerful and easier to use computer languages, plus various design methodologies have all helped in the battle for higher quality software. However a huge backlog of computer applications still exist within the industry. Some reports suggest that the backlog is so great that it deters managers from even suggesting other development projects. It is only now with the prominence of Object Oriented technology, that a serious attempt to solve this software backlog problem can be made.

Problems related to maintenance, complexity, software quality etc, can be tackled in a more efficient and reliable manner using the encapsulation, inheritance and data abstraction techniques promoted by most Object Oriented languages. It would be difficult for the software industry to progress in a manner similar to the hardware industry where engineers develop from existing circuitry. But by using Object Oriented language for reusing and extending existing software, the speed currently associated with development can be greatly increased. Before achieving these goals however, a number of improvements to Object Oriented technology will have to occur. This chapter takes a brief look into the future at some of the important features that must be obtained if the approach is to be an important tool for system development. As well as discussing the needed improvements to Object Oriented mechanisms, enhancements to the workbench and the Objective-C language are also discussed.

8.2 Future Enhancements to Objective-C

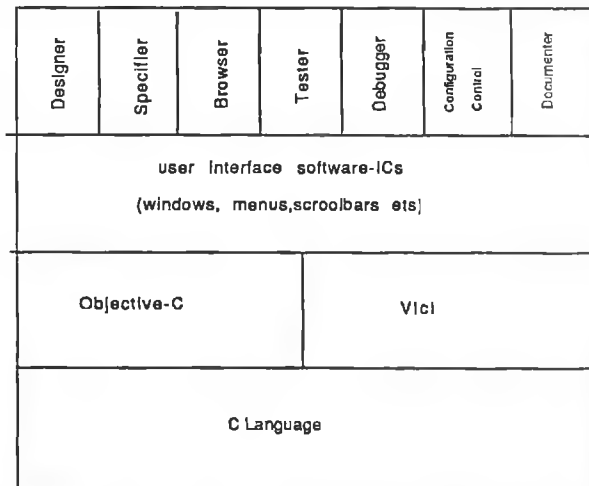
The StepStone Corporation, the suppliers of Objective-C, which was used for the development of tools in the workbench, expect their language to evolve in a manner similar to any factory object created by the language (i.e. additional features are added in an incremental fashion without effecting existing objects). While providing a comprehensive foundation library and "VICI", an interpreter for allowing code to be examined line by line, there are still areas where the language must improve. One of the major drawbacks which the supplier must investigate is automatic garbage collection i.e. the freeing of object space from memory once an object has been deleted. Developing systems using the current language version(3.3) puts the emphasis on the developer to follow an object's lifecycle from its creation to its deletion. Learning about an object's lifecycle puts an onus on the system developer to acquire a much greater in-depth knowledge of the overall system than should really be required. Having to re-learn about existing classes already validated and entered into the class hierarchy, goes against the reusability and extendibility issues associated with Object Oriented languages, this re-learning could be considered to be maintenance or additional work.

The suppliers of Objective-C recognise this problem plus others which must be solved if the language is to gain widespread commercial acceptance(i.e. the need for tools to help with design, debugging, documentation and testing of Object Oriented applications). The diagram in figure 8.1 shows the type of ideal development environment required for using the Objective-C language, followed by a description of the functionality expected from each tool.

VICI is an interpreter supplied along with the Objective-C compiler, which provides extensive trace and help facilities for both Objective-C and C. Using VICI allows development to be made without the compile link process between tests.

An environment similar to this should be the goal for all Object Oriented languages, not just Objective-C. The features in figure 8.1 represent the most desirable tools Object Oriented developers would wish for when developing systems using an Object Oriented environment.

The more immediate future of the language will improve, as Objective-C becomes available on a wider range of machines, such as Sun, Apollo, Hewlett Packard workstations and top range IBM personal computers. The recent decision by NExT to include Objective-C with their new workstations, strengthens the view, that Objective-C will remain as one of the main Object Oriented lan-



- **Designer** Tool to help the design process. It uses graphic windows to show the objects required for development and how they should be linked into the class hierarchy.
- **Debugger** Something similar to the dbx tool supplied by the Unix operating system. The interface however should be more graphic making code debugging easier.
- **Tester** Tool to generate tests on the object classes.
- **Documenter** Tool which generates standard documentation for each factory object.
- **Specifier** Tool which allows the user to describe a system graphically with verbal representations.
- **Browser** similar to the Smalltalk tool used to guide the user through the inheritance hierarchy.
- **Configuration Control** Used for synchronising the thread of control through an application.

Figure 8.1: Ideal Objective-C environment

guages for quite some time to come.

8.3 Future advancements to the Workbench

The architecture used for the creation of the workbench was developed using the IDEF-SADT approach. The various modules used the underlying principles of Object Oriented technology for development whenever possible. The three main components which form the workbench can be considered as objects in the Object Oriented sense. It would be incorrect however to say that these were completely reusable objects. The information system(IS) for example would require numerous changes to reference different object types and new rules for verifying these object types.

The computational system(CS) which contains the software tools for the IMPW provides better Object Oriented characteristics allowing the addition and extension of the software tools. The fact that each tool can be regarded as an object with well-defined boundaries, permits the changing and addition of tools with minimal effect to the surrounding workbench tools. As the workbench matures, work required on any tool whether to make it more reliable, faster, more comprehensible etc, can be performed in isolation of the other tools.

Because the current version of the workbench has a prototype label attached to it, a number of shortcomings were overlooked during development. From personal experience of using the workbench, one of the most immediate problems which would need to be solved is the speeding up operations on the data base. Retrieving and storing data from the IS database, has time delays which would be unacceptable to project managers in the every day industrial environment. Perhaps using a more efficient database, compiling the Prolog layer from the workbench IS, are two areas which could be investigated further, if the product is to be speeded up to an acceptable level.

Another flaw with the current workbench version is that it only permits one tool to be active during execution. In the real world environment software managers may wish to implement a number of tools simultaneously. For example, the project manager may wish to view the Risk and Calendar tools while using the Resource Allocation tool. In order to make the workbench more acceptable in the commercial marketplace, this transaction from sequential to parallel processing must be made.

The prototype workbench developed currently runs on Apollo and Sun

workstations. Extending this portability into other workstations and top range PCs must be considered for future versions, before the product can become commercially viable. The encapsulation principle allows objects used throughout the IMPW to reduce the work in such a transition. The porting of the greatest percentage of classes should just be a matter of recompilation.

8.4 User Interfaces

The area of computing where many people believe the real benefits of Object Oriented will be realised is in the area of computer interfaces. Presenting information in a graphical format which permits interaction in a way which is simple to learn and comprehend. Operations such as manipulation of objects representing real world entities by pressing a mouse button, can become the norm rather than the exception for computer users. The hardware and software capabilities have been around for some time, to build interactive systems like the one just mentioned. However the reasons why they are so sparse, is due to the complexity associated with developing such applications.

The introduction of Object Oriented languages with inheritance, can break down these technical barriers by allowing complex graphic code to be inherited. Creating graphic windows should be simply a matter of inheriting the correct graphic classes, leaving only minimal coding for the users who can tailor the displays to their own personal taste. The diagram in figure 8.2 shows how the inheritance mechanism could be used for the construction of such a system.

As mentioned earlier, no standard approach for the construction of classes exists, therefore the organisation of classes in this diagram should not be taken as standard. Some languages may take the viewpoint that the windows are rectangular boxes and therefore make the Window class the parent of the Rectangle class. This however should be regarded as incorrect classification of object classes. By making window the parent class, it would obstruct the Rectangle class being used in other applications such as mathematics.

This example shows a single example of how non-standardisation of class inheritance can effect the windowing system. However the side affects caused by such inheritance will not always be as easy to follow. If we wish to build flexible and correct window systems, consistency between the various languages must be obtained.

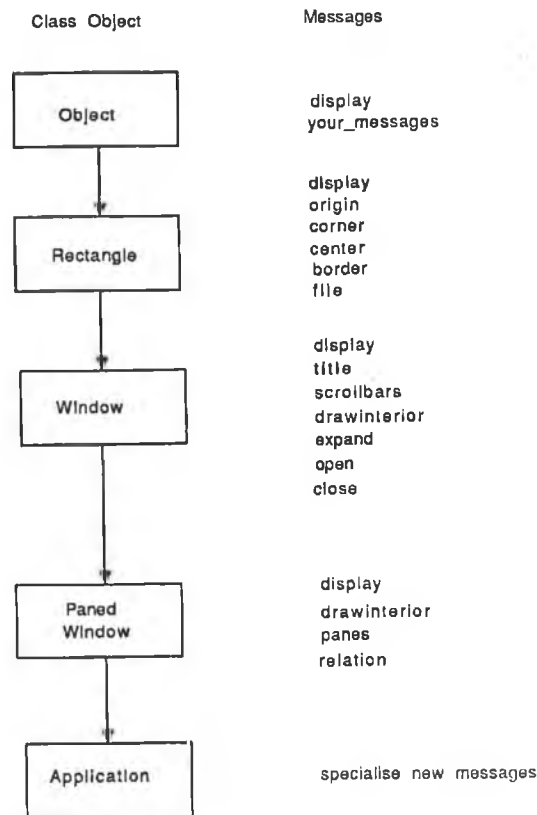


Figure 8.2: Possible Window Hierarchy

8.5 Conventional Systems

With all these great promises offered by Object Oriented technology, what will happen systems developed using conventional languages and methodologies? Should schools start teaching students development using only the Object Oriented approach? Should all new development take place in an Object Oriented environment? With the Object Oriented techniques still in their infancy, and the amount of investment currently made in traditional systems, work using conventional methods will persist for some considerable time. People are not going to throw away their tried and trusted systems just to keep up with computer technology. The time and money spent maintaining these conventional applications will in many cases be justified continually by someone in a senior position, convincing themselves that the changes are small and once off. There will also be systems where the amount of investment is so great, that changes will be made

rather than scrapping the existing system.

Other factors beside the economic reasons for keeping conventional systems include psychological and familiarity problems. Code which has not been developed in house is often regarded by people within the organisation as being erroneous. The familiarity problem relates to the fact most of the computer professionals have been brought up with a different concept towards system building using conventional approaches such as Jackson Structured approach. Hardened computer personnel used to certain techniques for a number of years, may try and resist the change, in the way office workers resisted the introduction of the computer into their work environment. However with the introduction of language environments as shown in figure 8.1 this problem can be overcome.

8.6 Object Oriented languages

The emergence of Object Oriented languages offer a number of new types and mechanism for the computer professional to learn. For the newcomer terminology about classes, instances, types, instance variables etc, can lead to more confusion rather than simplification. With no standards set, it is common to have different naming conventions to represent the same thing. For example type and factory may both refer to an object class, depending on the language used.

Indeed a more controversial and fundamental argument still exists between various professionals as to what exactly is an object. Database, Interface and AI developers using Object Oriented languages all have their own fuzzy notion as to what exactly an object is.

Standardisation of these Object Oriented principles must be regarded as an important characteristic if the goals mentioned at the start of this text are to be obtained. Throughout this chapter, numerous references have been made regarding consistency and standards; the three areas listed below indicate where setting of standards are required most.

- naming conventions
- class library structures
- design

8.6.1 Naming Conventions

The standardisation of naming conventions is not dedicated to having standard names to describe the characteristics introduced by Object Oriented principles, mentioned previously. Instead it is directed more towards message calling protocol. Analogous objects which exist in various languages use different message protocols for calling similar operations. In a more ideal Object Oriented environment, similar objects in different languages should have the same class name and associated operations so that one message protocol could exist for all Object Oriented languages. Going back to our window example, if different languages use different commands to perform similar operations (i.e. draw and display methods both ask an object to display itself) system development can only proceed within the boundaries of one language. If Object Oriented suppliers seriously want to encourage reuse of software, they should keep the messages and operations related to a class consistent between languages. By doing this developers will not have to learn any extra syntax for messages when using different Object Oriented language.

8.6.2 Class Library Structures

The different Object Oriented language suppliers not only supply different classes, but also different inheritance mechanism structures. The position of certain object classes is dependent on the language i.e. Smalltalk and Objective-C use different classes and hierarchical structures for storing and retrieving these library objects. As the interest in Object Oriented programming grows, and suppliers jostle for a premier position in this new market place, the trend of most suppliers is to keep putting more reusable classes into their language library, instead of looking at what has been developed by other similar suppliers. Getting the various suppliers to talk about a standard hierarchy for object classes and their operations, is an important step towards standardisation of Object Oriented languages.

8.6.3 Design

Object Oriented Design is a research area which requires a great amount of improvement. The introduction of inheritance, encapsulation and polymorphism characteristics from Object Oriented languages is poorly supported by current design methodologies. Most of the current approaches for this technology are

directed towards the development of systems implemented in Ada. The fact that Ada does not have class inheritance makes it a poor language for modelling Object Oriented systems[Rob 81],[Weg 87][Ver 88]. In fact for this reason many Object Oriented personnel do not consider Ada to be a proper Object Oriented language.

Having reusable and extendible design components which could be fitted into various applications in a manner similar to object classes, is the design goal which the Object Oriented approach is striving for. Changing software design from its art/craftmanship status to an industrial process, where design components are obtained in the same way as we purchase integrated circuits is however a long way from reality. Framework models which support this type of reusability have already been developed[Jac 87]. However this approach for model creation was developed for large software projects (i.e. those greater than ten man years, involving numerous personnel). While promoting reusability in large software project, the model was developed for implementation within the companies organisation boundaries. The various design components and steps involved in this approach make its use in smaller applications and in different environments questionable.

Different inheritance structures talked about previously, cause interference which makes it impossible for the design not to be influenced by the language. If the designer wishes to include inheritance in the design, then object classes must be attached to a known language inheritance structure. Therefore Object Oriented design i.e. include encapsulation and inheritance, implies that the design rule stating that "design should be independent of language" should be ignored.

Because of the differences in language libraries, design is affected by the Object Oriented language used for implementation. Object Oriented languages such as Objective-C, C++, Eiffel etc, all contain different classes, class inheritance mechanisms, and operations on their classes. Clearly there is a need for developing standards so that objects created by different suppliers can be glued together for system development.

This design problem only reinforces the call to set standards for inheritance, classes and the operations performed by the classes. However this problem cannot be considered trivial, maintaining standards for Object Oriented design and implementation introduces problems of its own. To allow the addition of further objects and the evolution of current classes, stringent management plus global communication channels are required.

Deciding what classes to enter should be dependent mainly on whether

the classes are reusable for other applications. As mentioned in chapter seven designing these abstract classes for multiple use is difficult. The main method used for creating such reusable classes was experience. This highlights the need for a tool which could help with the creation of reusable classes, giving details of the operations, data and inheritance required .

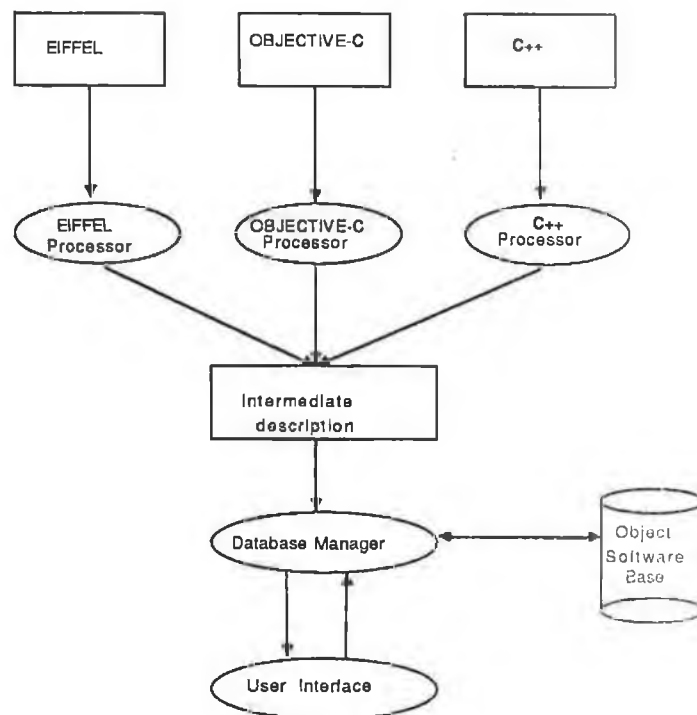


Figure 8.3: Object Database Hierarchy

A DBMS can be used to perform an important role in helping to achieve this objective. Research into storing object classes inside data base environments has already commenced. The entity relationship modelling represent a more natural way to represent the relationships between objects. The diagram in figure 8.3 shows the architecture of an object database currently being developed at Geneva University[Ara 88]. Such an environment is one possible solution to the problem of managing object classes supplied by different Object Oriented language suppliers.

8.7 Summary

The benefits that can be achieved from reusable and extendible code can catapult the Object Oriented approach to the forefront of computer technology. Fifth generation computers, Artificial Intelligence, and CAD/CAM systems can all benefit by adopting the Object Oriented approach. Reusable code will mean higher quality software and better documentation than before. System extensions can be performed in an incremental fashion without affecting existing code, thanks to encapsulation. However before reaching this stage a standard design methodology which can be applied to all Object Oriented languages is required. This in turn implies that standards are established for the various languages.

As the approach becomes more popular, better information on design and implementation issues should become available. Books and journals on the subject have increased significantly over the past few years to help newcomers get to grips with this new approach. There are also a number of Object Oriented conferences which present the most recent innovations in this area; OOPSLA and ECOOP are two of the most popular ones. Newcomers can also learn by going to intensive training course which provide an excellent introduction for understanding and implementing systems which use this approach. Computer professionals however should not feel threatened, creativity and clear headed thinking will still be the most important ingredients for system design and implementation.

Appendix A

Risk Management Areas

A definition of the four 'Risk Management Areas'.

The term "Risk Management Area" is drawn from current literature on software project risk management. The four Risk Management Areas chosen for the prototype tool were:

Development Risk 1: Cost/Schedule Failure

"The risk that project cost and/or duration may significantly exceed best point (ie. preferred) estimate."

Development Risk 2: Premature Project Termination

"The risk that the project may have to be prematurely terminated or radically rescope/revised because of major technical or resource problems which might arise."

Operational Risk 1: Product Functional Failure

"The risk that the functionality in the delivered product may fail to meet the client's expectations/needs."

In essence, this is the risk of making the "wrong" product.

Operational Risk 2: Product Technical Failure

"The risk that the product may fail to work in the target environment for technical reasons (eg. because of failure to interface with other components)."

Appendix B

Risk Drivers

A Definition of the "Risk Drivers".

The rating scales presented to the manager are shown below. The manager selects his responses with the mouse. For this appendix, the format of presentation of the scales has been squeezed up somewhat. Towards the right-hand margin of each question is shown a scale of integer values. These are the "risk points" assigned to each possible response. These are not displayed to the manager. The use of the risk points by the tool is explained in Appendix C. A text paragraph appears under most scales. This is to help the manager select his response.

1. THE CLIENT'S UNDERSTANDING OF HIS REQUIREMENTS

The client has almost no understanding of his requirements	[]	4
The client has some understanding of his requirements	[]	3
The client has quite a good understanding of his requirements	[]	2
The client has an excellent understanding of his requirements	[]	1

(By "understanding" is meant the ability of the client to accurately percieve and articulate the product requirements)

2. THE DESIGNERS' KNOWLEDGE OF THE APPLICATION DOMAIN

The designers have an excellent knowledge of the application domain	[]	1
The designers have a good knowledge of the application domain	[]	2
The designers have just a little knowledge of the application domain	[]	3
The designers have no knowledge of the application domain	[]	4

(By "good knowledge" is meant an exposure to ,for example, the practical constraints in the user's environment that the product must work within or the possession of insight into the "real " requirements from a functionality point of view. This knowledge might have been gained by working as a user or analyst in the area.)

3. AVAILABILITY OF EXISTING PRODUCTS (OR PROTOTYPES) WHICH CAN ACT AS EXAMPLES FOR THE DESIGNERS

The designers can refer to other products (or prototypes) having functionality which is- identical to the required functionality	[]	1
very similar to the required functionality	[]	2
somewhat similar to the required functionality	[]	3
no similar example is available	[]	4

4. EXPERIENCE OF TEAM MEMBERS IN THE TECHNICAL TASKS OF DEVELOPING SOFTWARE FOR THIS APPLICATION DOMAIN

Nobody on the team has good experience	[]	4
A small proportion have good experience	[]	3
A large proportion have good experience	[]	2
Most or all have good experience	[]	1

(By "experience" is meant previous exposure that will help the team to anticipate and solve application-specific technical problems.)

5. THE NEED TO SOLVE VERY DIFFICULT TECHNICAL OR INTELLECTUAL PROBLEMS AS PART OF THE PROJECT

The success of this project depends on our solving-

very difficult technical/intellectual problems	<input type="checkbox"/>	4
moderately difficult technical/intellectual problems	<input type="checkbox"/>	3
fairly easy technical/intellectual problems	<input type="checkbox"/>	2
only very easy technical/intellectual problems	<input type="checkbox"/>	1

(By "very difficult technical/intellectual problems" are meant original problems which may turn out to be unsolvable ,or the solving of which cannot be guaranteed within a given timescale, irrespective of the resources devoted to their solution.)

6. POSSIBILITY OF TESTING THE PRODUCT IN A "SAFE" ENVIRONMENT

Will it be possible to test the product (or prototypes of the product) in a "safe" environment which is representative of the final use environment?

yes	<input type="checkbox"/>	1
no	<input type="checkbox"/>	4

7. "SIZE" OF THE PRODUCT

In relation to what we are accustomed, the product is-

very small or is easily broken down into normal-size work packages	<input type="checkbox"/>	1
fairly small or fairly easily broken down into normal-size work packages	<input type="checkbox"/>	2
fairly large or not easily broken down into normal-size work packages	<input type="checkbox"/>	3
very large or cannot be broken down into normal-size work packages	<input type="checkbox"/>	4

8. COMPLEXITY OF PRODUCT REQUIREMENTS

In relation to what we are accustomed-

Requirements are very simple and easily allocated to software components/modules	<input type="checkbox"/>	1
Requirements are fairly simple and easily allocated to software components/modules	<input type="checkbox"/>	2
Requirements are fairly complex and not easily allocated to software components/modules	<input type="checkbox"/>	3
Requirements are very complex and can be allocated to software components/modules only with great difficulty	<input type="checkbox"/>	4

9. LEVEL OF VOLATILITY OF PRODUCT REQUIREMENTS DURING THE PROJECT

During the course of development , product requirements are likely to be subject to-

very extensive revision	<input type="checkbox"/>	4
extensive revision	<input type="checkbox"/>	3
some revision	<input type="checkbox"/>	2
little or no revision	<input type="checkbox"/>	1

10. STABILITY OF OPERATIONAL INTERFACES

Interfaces between the product and other software and hardware components it must work with in the final use environment are-

very well defined and subject only to tightly controlled change	<input type="checkbox"/>	1
quite well defined and subject only to tightly controlled change	<input type="checkbox"/>	2
badly-defined or subject to uncontrolled change	<input type="checkbox"/>	4

11. INFLEXIBILITY OF FUNCTIONAL AND OTHER SPECIFICATIONS

If we meet problems in development, it will be-

Impossible to agree changes to functional and other specifications	<input type="checkbox"/>	4
Very difficult to agree changes to functional and other specifications	<input type="checkbox"/>	3
Difficult to agree changes to functional and other specifications	<input type="checkbox"/>	2
Not too difficult to agree changes to functional and other specifications	<input type="checkbox"/>	1

12. SCALE OF PROJECT (NO. OF PEOPLE)

In relation to what we are accustomed, the size of the project team is-

at least three times as big	<input type="checkbox"/>	4
about twice as big	<input type="checkbox"/>	3
about the same size	<input type="checkbox"/>	2
smaller	<input type="checkbox"/>	1

13. SCALE OF PROJECT (DURATION)

In relation to what we are accustomed, the duration of the project is likely to be-

at least three times the length	<input type="checkbox"/>	4
about twice the length	<input type="checkbox"/>	3
about the same length	<input type="checkbox"/>	2
shorter	<input type="checkbox"/>	1

14. MATURITY OF THE DEVELOPMENT ENVIRONMENT

The development environment to be used is-

very novel/untested	<input type="checkbox"/>	4
fairly novel/untested	<input type="checkbox"/>	3
fairly mature/tested	<input type="checkbox"/>	2
very mature/tested	<input type="checkbox"/>	1

(By "development environment" is meant the software tools, languages, methods, hardware etc. to be used in development.)

15. EXPERIENCE OF THE DEVELOPERS WITH THE DEVELOPMENT ENVIRONMENT

With regard to experience of the development environment to be used, the team contains-

no experience	<input type="checkbox"/>	4
a little experience	<input type="checkbox"/>	3
quite a lot of experience	<input type="checkbox"/>	2
extensive experience	<input type="checkbox"/>	1

16. MATURITY OF THE TECHNICAL TARGET ENVIRONMENT

The target environment is -

very novel/untested	<input type="checkbox"/>	4
fairly novel/untested	<input type="checkbox"/>	3
fairly mature/tested	<input type="checkbox"/>	2
very mature/tested	<input type="checkbox"/>	1

(By "technical target environment" is meant the hardware/software environment that the product is to run in.)

17. EXPERIENCE OF TEAM MEMBERS OF THE TECHNICAL TARGET ENVIRONMENT

Nobody on the team has good experience	<input type="checkbox"/>	4
Only a small proportion have good experience	<input type="checkbox"/>	3
A significant proportion have good experience	<input type="checkbox"/>	2
Most or all have good experience	<input type="checkbox"/>	1

(By "good experience" is meant a sufficient exposure to the target environment (eg. operating system, tp monitor) to be

able to anticipate / solve technical problems.)

18. THE COMPLEXITY OF COMMUNICATION LINKAGES WITH ANY COLLABORATORS OR SUBCONTRACTORS		
Communication linkages with any collaborators or subcontractors are-		
highly complex	<input type="checkbox"/>	4
quite complex	<input type="checkbox"/>	2
not complex	<input type="checkbox"/>	1

("complexity" of communication linkages refers to the number of collaborators or subcontractors involved, the number of contact points with each , to problems due to geography or language , to a need to cope with conflict or politics etc.)

19. THE COMPLEXITY OF COMMUNICATION LINKAGES WITH THE CLIENT		
Communication linkages with the client are-		
highly complex	<input type="checkbox"/>	4
quite complex	<input type="checkbox"/>	2
not complex	<input type="checkbox"/>	1

("complexity" of communication linkages refers to the number of contact points between the project and the client, to problems due to geography or language , to a need to cope with conflict or politics in the client organisation etc.)

20. VOLATILITY OF MEMBERSHIP OF THE PROJECT-TEAM		
During the course of the project , turnover of team-membership will probably be-		
three-quarters or more	<input type="checkbox"/>	4
between a half and three quarters	<input type="checkbox"/>	3
between a quarter and a half	<input type="checkbox"/>	2
less than a quarter	<input type="checkbox"/>	1

("team-membership" refers to people who have a significant role in developing the product. Support staff or others who might have a less central role should be excluded.)

21. RISK OF LOSS OF MOST IMPORTANT TEAM MEMBERS		
The loss of one or more critical team-members during the project is-		
very likely	<input type="checkbox"/>	4
likely	<input type="checkbox"/>	3
unlikely	<input type="checkbox"/>	2
very unlikely	<input type="checkbox"/>	1

(A "critical team-member " is someone whose departure could badly disrupt progress or even lead to termination of the project .)

22. PROJECT MANAGER'S LEVEL OF KNOWLEDGE OF THE SKILLS AND PRODUCTIVITY OF TEAM-MEMBERS		
The project manager has a good knowledge of the skills and productivity of-		
less than a quarter of team-members	<input type="checkbox"/>	4
between a quarter and a half of team members	<input type="checkbox"/>	3
between half and three-quarters of team-members	<input type="checkbox"/>	2
more than three-quarters of team-members	<input type="checkbox"/>	1

("team-members" refers to people who have a significant role in developing the product. Support staff or others who might have a less central role should be excluded.)

23. LEVEL OF DEPENDENCE OF THE PROJECT ON "RISKY" IMPORTS		
The project will be-		
critically dependent on risky imports	<input type="checkbox"/>	4
highly dependent on risky imports	<input type="checkbox"/>	3
somewhat dependent on risky imports	<input type="checkbox"/>	2

not dependent on risky imports [] 1

(Examples of imports would include re-usable software, equipment and tools, buildings, important people etc. Risk refers to the possibility that the import might not be available when required or might not be suitable for its purpose eg. be of poor quality, be incompatible etc.)

Appendix C

Risk Measures

Algorithm for computing the four Risk Measures.

The four Risk Measures (one for each Risk Management Area) are computed as weighted linear functions of the values of the Risk Drivers. Each Risk Driver may contribute to one or more of the four Risk Measures. The "mapping" of Risk Drivers onto the Risk Measures is shown below. In other words, the diagram shows which Risk Drivers are deemed to contribute to each Risk Measure (* indicates a contribution).

RISK DRIVER	DEVELOPMENT RISK 1	DEVELOPMENT RISK 2	OPERATIONAL RISK 1	OPERATIONAL RISK 2
1	*		*	
2	*		*	
3	*		*	*
4	*	*		*
5	*	*		
6			*	*
7	*			
8	*			
9	*		*	
10	*			*
11	*	*		
12	*			
13	*			
14	*	*		
15	*	*		
16	*	*		*
17	*	*		*
18	*			
19	*		*	
20	*			
21	*	*		
22	*			
23	*	*		

A positive integer (ie. number of "Risk Points") is assigned to each of the possible responses to each of the Risk Driver scales. The initial assignment of points has been made by the tool developers. This initial assignment of points may be amended by the system manager using the "Amend Risk Driver" function. Unavoidably, the allocation of points to each possible response is a subjective process, but the points assigned have been constrained to form a monotonic scale within the Risk Driver which rises as the implied "riskiness" of the manager's response rises. The Risk Measure for a Risk Management Area is computed as the sum of the points associated with the manager's responses on the Risk Drivers contributing to that Risk Management Area, expressed as a percentage of the maximum sum of points achievable for that Risk Management Area.

It is recognised that this is a somewhat subjective scoring method. If an alternative with better properties is found, it will be adopted.

A provisional assignment of "risk points" has been made to each possible response on each Risk Driver. The risk points are shown in Appendix B.

Appendix D

Risk Tool Classes

```

// Objective_C main program module
// The purpose of this program is to allow two different types of users
// to add information to the Risk Analysis Tool. The two types of users
// are :- 1. The administration user (a large percentage of the
//         functional modules in this system are for this user).
//         2. The end user (enters project characteristics prompted
//         by a menu driven system).
//
//         RISK ANALYSIS TOOL

#include "sac_global.h"
#include "main.h"
#include <stdio.h>
#include <objc.h>
#include "risk.h"

@requires RiskAutomata;
@requires OrdCltn ;
@requires IS;

@requires AsciiFiler, String, Sequence, RkCltn, Interface ;
@requires Risk, Rule, Txt, User, Measure, ProdDef, Graphic ;

id anAutomata;
extern BOOL msgFlag ;

extern int errno;
extern FILE *yyin;
char *malloc() ;

id tempprod ;

/* graphical screen */
Gevscrgh_t screen;

/* window */
Gevwdw_t    window, window1;
Gevwdw_t    currentwdw;

Gevlift_t   voidlift;
Rpos_t      p1, p2, p3    ;
Gevicocar_t voidico      ;

/* row */
Gevrow_t     risktitle, risktextual;
Gevrow_t     risk_rows, top, topright;
Gevrow_t     riskdetails, conditions, condition;

/* icon table */
Gevtab_t     someobject;

/* content */
Matrix_t     confirm_hd_mat ;
Matrix_t     confirm_mat ;

Gevspa_t     Space, boxedspace;
Gevspa_t     boxedspace1, boxedspace2, boxedspace3, boxedspace4;
Gevspa_t     boxedspace5 ;

Gevspa_t     boxed1, boxed2, boxed3, boxed4;
Gevspa_t     boxed5 ;

/* icon character */
Gevicocar_t  topleft, projnam, projtitle;

```

```

Gevicocar_t  titleico;
Gevicocar_t  rdico, rulico, ameico, amaico, repico ;
Gevicocar_t  admico, userico ;
Gevicocar_t  confirmicoheader, aborticoheader;
Gevicocar_t  confirmico, abortico;
Gevicocar_t  partofline;
Gevicocar_t  riskhelpico ;

Gevrow_t      error_msg_row;
Gevmsg_t      error_msg;

Gevlift_t     lifthor, liftver ;

/* global variable */
Pos_t         position;
Siz_t         taille;
Rpos_t        rposition;
Rsiz_t        rtaille;
Gevevt_t      evt;
Matrix_t      voidmatrix ;
int           endofjob;

=(RiskGroup, Primitive, Collection)

main( argc , argv, arge )
    int argc;
    char * argv[];
    char * arge[] ;
{
    id base;

    /* MWI initialization call */
    str_intern(&_outil, "impw");
    env_init();

    if (strcmp(argv[2], "") != 0) {
        base = [String str: "database "];
        [base concatSTR: argv[2]];
    }
    else {
        printf("You must enter the database name\n");
        [IS interpret: "quit"];
    }

    create_fifo();

    [IS interpret: [base str]];

    // 1 - debug trace for message calls
    if (argc > 1 && *argv[1] == 't')
        msgFlag = YES ;
    [RkCltn startUp] ;    // Enters Riskfactors and Rules from disk if
        // saved and required.
    [Graphic create];
    [ProdDef initialise] ;
    tempprod = [ProdDef add] ;
    // [tempprod print] ;
    // 3 - if none saved automata state exist creates
    //      a new one and lunch it
    if (anAutomata == nil){
        anAutomata = [RiskAutomata new];
        [anAutomata initialise];
        [anAutomata execute: nil];
    }
}

```



```

        else
            // lunch the saved automata state
            [anAutomata execute: nil];
    }

create_fifo()
{
    char *self_fifo = "/tmp/self_fifoXXXXXX", *mktemp();
    int d;

    mknod((self_fifo = mktemp(self_fifo)), 0010600, 0);
    d = open(self_fifo, O_RDWR|O_NDELAY, 0);

    close(d);
    yyin = fopen(self_fifo, "r+");
}

@classes()
@messages()

#include <stdio.h>
#include <objc.h>
#include <math.h>
#include "Riskmess.h"
#include "sac_global.h"

@requires OrdCltn;
@requires Interface;
@requires Rule;
@requires Measure;
@requires RkCltn;
@requires Graphic;
@requires String;
@requires IS;

extern id tempprod;

extern riskCltn, txtCltn, ruleCltn;
extern char *operand[10], *operator[10];
extern char *iobuf;
extern BOOL VALID;

extern Matrix_t      confirm_mat;
extern Matrix_t      voidmatrix;

extern Gevlift_t      lifthor, liftver;

extern Gevtab_t       confirm_tab;

extern Gevicocar_t    quitRiskDriverViewIcon;
extern Gevicocar_t    notitle;
extern Gevicocar_t    riskhelpico;
extern Gevicocar_t    partoffline;
extern Gevicocar_t    voidico;
extern Gevicocar_t    confirmicoheader, aborticoheader;
extern Gevicocar_t    confirmico, abortico;

extern Gevrow_t       risk_rows , top;
extern Gevrow_t       riskdetails, conditions, condition;

extern Gevwdw_t       window, window1, currentwdw;

extern Gevscrph_t     screen;

extern Siz_t          taille;

```

```

extern Pos_t          position;
extern Rpos_t         p1, p2, p3;

extern Gevspa_t       Space, boxedspace ;
extern Gevspa_t       boxed1, boxed2, boxed3, boxed4, boxed5;
extern Gevspa_t       boxedspace1, boxedspace2, boxedspace3;
extern Gevspa_t       boxedspace4, boxedspace5;
extern Gevrow_t       risktitle, risktextual, someobject;
extern Gevrow_t       error_msg_row;
extern Gevlift_t      voidlift;
extern Gevmmsg_t      error_msg;
extern char *         proj_nam;

Gevicocar_t          bottom;
Gevicocar_t          riskdriverico, risknumico;
Gevrow_t             risknum_row;
Gevchr_t             risknum;
Gevtab_t             risktoptab;
Gevvt_t             evt;
Matrix_t            risk_mat;

int      endofjob;
int  INDEX = 0;
char *tmpStr[20];

static int Risknum;
#define max(A,B) ((A) > (B) ? (A) : (B))

// Objective-C source file for the class risk
= Risk : Object ( RiskGroup, Collection, Primitive )
{
    char *riskdriver ;
    char *risktxt ;
    char *riskcondition[6] ;
    int  riskweight[6] ;
    char *riskhlp ;
    char *attrName;
    char *entityName;
}

char *copycat(), *malloc() ;

//      FACTORY METHODS

+ create {      // Create a new Riskdriver for insertion
    id risknum ; // onto Risk Analysis Tool.
    int i = 0 ;
    int weight ;

    self = [super new] ;
    system("clear") ;
    printf("\n\n\n\n\tEnter Riskdriver title ") ; //Riskdriver name
    copycat() ;
    riskdriver = malloc(strlen(iobuf)) ;
    strcpy(riskdriver,iobuf) ;
    printf("\n\n\n\n\tEnter RiskDriver text :- ") ;
    copycat() ;
    risktxt = malloc(strlen(iobuf)) ;
    strcpy(risktxt,iobuf) ;
    do {      // enter Riskdriver conditions until empty line
        printf("\n\tEnter risk condition %d ",i+1) ;
        copycat() ;
        riskcondition[i] = malloc(strlen(iobuf)) ;
        strcpy(riskcondition[i],iobuf) ;
        if (strlen(iobuf) < 1)
        {
            riskcondition[i] = malloc(12) ;
            strcpy(riskcondition[i],"don't know") ;
            riskweight[i] = 0 ; }
    }

```

```

else
{
    printf("\n\t%d\tEnter risk weight ",i+1) ;
    scanf("%d",&weight) ;
    riskweight[i] = weight ; }
} while (strcmp(riskcondition[i++],"don't know") != 0) ;
printf("\n\n\tEnter riskdriver screen help :- ") ;
strcpycat() ;
riskhlp = malloc(strlen(iobuf)) ;
strcpy(riskhlp,iobuf) ;
risknum = [Measure create] ; // defines the riskareas which
[risknum riskarea:[riskCltn size]] ;// the Riskdriver will be
[Measure addriskAreas:risknum] ; // associated with.
return self ;
}

// This method deletes a Riskdriver checking
// if it corresponds to a Rule in the RuleBase
+ delete {
    id aRisk,aRule ;
    char *tmpChar, *tmpCond, cont, confirm ;
    int num, tmp, len ;
    int i, k,i1, k1 ;
    int delarray[10] ;
    BOOL FOUND, FOUND1 ;
    id delCltn, delSeq ;

    tmpCond = malloc(8) ;
    delCltn = [OrdCltn new:10] ;
    do {
num = [Risk getnum] ;
if (num != 0) // num is inside the range of the Risk database display
{
    // the Riskdriver the user is wishing to delete
    aRisk = [[riskCltn at:--num] print] ; // Display the Rule which
    // will be effected if this Riskdriver is removed.
    i = 0 ; num++ ;
    len = [ruleCltn size] ;
    while (i < len) { // more rows to read
        FOUND = NO ;
        k = 0 ;
        aRule = [ruleCltn at:i++] ;
        tmpChar = [aRule rules:k] ;
        while (*tmpChar != '*' && !FOUND) {
            tmpChar = [aRule rules:k++] ;
            if (*tmpChar == 'R') {
                tmpChar++ ;
                tmp = atoi(tmpChar) ;
                if (tmp == num) { // this rule is affected by the
                    // deletion of the Riskdriver
                    [delCltn addIfAbsent:[ruleCltn at:i-1]] ;
                    strcpy(tmpCond,[aRule condition]) ;
                    while (i1 < len) { // more rows to read
                        FOUND1 = NO ;
                        k1 = 0 ;
                        aRule = [ruleCltn at:i1++] ;
                        tmpChar = [aRule rules:k1] ;
                        while (*tmpChar != '*' && !FOUND1) {
                            tmpChar = [aRule rules:k1++] ;
                            if (*tmpChar == 'C') {
                                tmpChar++ ;
                                if(strcmp(tmpCond,tmpChar) ==0) {
                                    [delCltn addIfAbsent:[ruleCltn at:i1-1]] ;
                                    FOUND1 = YES ;
                                }
                            }
                        }
                    }
                }
            }
        }
        i++ ;
    }
    FOUND = YES ;
}
}

```

```

    }
  }
}

// display the rules for deletion on screen
delSeq = [delCltn eachElement] ;
while (aRule = [delSeq next])
  [aRule print] ;
[DELETE_CONFIRM print] ;
scanf("%c",&confirm) ;
if (confirm == 'y' || confirm == 'Y') {
  delSeq = [delCltn eachElement] ;
  while (aRule = [delSeq next])
    [aRule removeRule:[ruleCltn offsetOf:aRule]] ;
    [Rule update:num] ;
    [riskCltn remove:aRisk] ;
}
} // end else
[DELETE_CONT print] ;
scanf("%c",&cont) ;
} while (cont == 'y' || cont == 'Y') ;
return self ;
} // end delete method

// Creates a new copy of the instance you wish to amend, changes are
// made on the copy which overwrites the riskdriver receiver in the
// ruleCltn if required.
+ amend {
  id aRisk, bRisk ;
  char ans, ans1, cont ;
  int num ;

  do { system("clear") ;
    printf("\n\n\n\t") ;
    num = [Risk getnum] ;
    if (num != 0) {
      bRisk = [riskCltn at:--num];
      aRisk = [Risk new] ;
      [aRisk copy:bRisk] ;
      printf("\n\n\n\t", [aRisk print]) ;
      [AMEND_THIS_RISKDRIVER print] ;
      scanf("%c",&ans) ;
      if (ans == 'y' || ans == 'Y') {
        [aRisk maintenance:num] ; // amendments made, ask user if
        // they wish overwrite riskdriver
        if ([aRisk notSame: bRisk]) {
          printf("\n\n\t Risk1 =") ;
          [bRisk print] ; // to update Riskdriver
          printf("\n\n\t Risk2 =") ;
          [aRisk print] ;
          [AMEND_COPY print] ;
          scanf("%c",&ans1) ;
          if (ans1 == 'y' || ans1 == 'Y')
            [[riskCltn insert:aRisk before:bRisk] remove: bRisk] ;
        }
      }
    }
    [AMEND_CONTINUE print] ;
    scanf("%c",&cont) ;
  } while (cont == 'y' || cont == 'Y') ;
  return self ;
}

```

```

/* ***** VIEW METHODS ***** */
/* ***** VIEW METHODS ***** */
/* Shows the private data of a Riskdriver
+ view {
    id aRisk ;

    [self get_num_risk_wdw];
do {
    Risknum = [Risk getnum] ;
    if (Risknum != 0) {
        aRisk = [riskCltn at:--Risknum];
        [aRisk print] ;
        [aRisk quit_risk_view]; }
    }
    while (Risknum != 0) ;
    return self ;
}

+ get_num_risk_wdw {

    risknum_row = Gevrow_create( GEV_VERTICAL, GEV_OUTLINE_OFF, GEV_SPACE_ON);

    risknumico = Gevicocar_create("Enter Riskdriver Number",GEV_FONT4,
        GEV_C, GEV_OUTLINE_OFF, GEV_SPACE_ON);

    risknum = Gevchr_create( " :- ",GEV_FONT4, GEV_FONT3,20,
        GEV_OUTLINE_OFF, GEV_SPACE_ON);

    error_msg_row = Gevrow_create(GEV_VERTICAL, GEV_OUTLINE_OFF,
        GEV_SPACE_ON);
    error_msg = Gevmsg_create("",GEV_FONT4,GEV_FONT4,18,
        GEV_OUTLINE_OFF,GEV_SPACE_ON);
    Gevrow_add_obj(error_msg_row,error_msg);

    Gevrow_add_obj(risknum_row,top);
    Gevrow_add_obj(risknum_row,Space);
    Gevrow_add_obj(risknum_row,risknumico);
    Gevrow_add_obj(risknum_row,risknum);
    Gevrow_add_obj(risknum_row,Space);

    Gevchr_deselect(confirmico);
    Gevchr_deselect(abortico);

    Gevrow_add_obj(risknum_row,confirm_tab);
    Gevrow_add_obj(risknum_row,error_msg_row);

    [self windowopen:risknum_row];
}

+ (char *) enter_risknum {
    int VALUE;
    char *numchr;

    while (!(Gevobj_eq(Gevvt_get_botobj(evt),risknum)) ||
        !(Gevobj_eq(Gevvt_get_ttyp(evt),GEV_VALIDATION)))
        evt = Gevscrgph_wait_event(screen);
    numchr = Gevchr_get_cont(risknum);

    while(!(Gevobj_eq(Gevvt_get_botobj(evt),confirmico)) ||
        !(Gevobj_eq(Gevvt_get_botobj(evt),abortico))) &&
        !(Gevobj_eq(Gevvt_get_ttyp(evt),GEV_SELECTED)))
        evt = Gevscrgph_wait_event(screen);

    if (Gevobj_eq(Gevvt_get_botobj(evt),abortico)) {

```

```

        Gevscrgph_rem_window(screen,currentwdw);
        numchr = ZEROCHAR;
    }

    if (numchr == 0)
        numchr = ZEROCHAR;
    return numchr;
}

// verifies the riskdriver number entered
+ (int) getnum {
    int num;
    int i;
    char *numStr, tmpStr[10];
    BOOL ERROR_FOUND;

    do {
        ERROR_FOUND = NO;
        numStr = [self enter_risknum];
        strcpy(tmpStr,numStr);
        i = 0;
        while ((tmpStr[i] != '\0') && !ERROR_FOUND) {
            if (tmpStr[i] < '0' || tmpStr[i] > '9') {
                ERROR_FOUND = YES;
                Gevmsg_set_cont(error_msg,"ERROR : non numeric value entered");
            }
            i++;
        }
        if (!ERROR_FOUND) {
            num = atoi(tmpStr);
            if (num < 0 || num > [riskCltn size])
                Gevmsg_set_cont(error_msg,"ERROR : invalid risk number");
        }
    } while (num < 0 || num > [riskCltn size]) ;
    Gevmsg_set_cont(error_msg,"");
    return num;
}

// print private data for riskdriver
- printOn: (IOD) anIOD {

    risk_rows = Gevrow_create(GEV_VERTICAL, GEV_OUTLINE_ON, GEV_SPACE_OFF);
    [self riskview];
    Gevrow_add_obj(risk_rows,quitRiskDriverViewIcon);
    window = Gevwdw_create(risk_rows);

    /* window belong graphic screen */
    Gevscrgph_add_window(screen, window);

    /* positioning and dimensioning in logical reference */
    Siz_set(taille, 500, 330);
    Pos_set(position, 600, 300);
    Gevwdw_set(window, position, taille);

    /* display of the window */
    Gevscrgph_display_window(screen, window);

    return self ;
}

- quit_risk_view {

```

```

    int i = 0;

    while (!(Gevobj_eq(Gevvt_get_botobj(evt),quitRiskDriverViewIcon) ||
        !(Gevobj_eq(Gevvt_get_ttyp(evt),GEV_BUTTON)))
        evt = Gevscrghph_wait_event(screen);
    Gevscrghph_rem_window(screen>window);
    while( i < INDEX )
        free(tmpStr[i++]);
    INDEX = 0;

    Gevchr_deselect(confirmico);
    Gevchr_deselect(abortico);

    return self;
}

- riskview {
    [self riskdriver_number];
    [self risk_title];
    [self risk_textual];
    [self risk_conditions];
    [self risk_help];
    return self;
}

- riskdriver_number {
    tmpStr[INDEX] = malloc(14);

    sprintf(tmpStr[INDEX]," Riskdriver %d",++Risknum);
    riskdriverico = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
        GEV_OUTLINE_OFF,GEV_SPACE_OFF);
    Gevrow_add_obj(risk_rows,riskdriverico);
    return self;
}

- risk_title {
    int i = 0;
    char sentence[6][65];
    sentence[0][0] = '\0' ;
    sentence[1][0] = '\0' ;
    sentence[2][0] = '\0' ;
    sentence[3][0] = '\0' ;
    sentence[4][0] = '\0' ;
    sentence[5][0] = '\0' ;

    textform([self riskdriver], sentence);
    while(sentence[i][0]) {
        tmpStr[INDEX] = malloc(strlen(sentence[i]) + 1);
        strcpy(tmpStr[INDEX],sentence[i++]);
        partoffline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
            GEV_OUTLINE_OFF,GEV_SPACE_OFF);
        Gevrow_add_obj(risk_rows,partoffline);
    }
    return self;
}

- risk_textual {
    int i = 0;
    char sen[6][65];
    sen[0][0] = '\0' ;
    sen[1][0] = '\0' ;
    sen[2][0] = '\0' ;
    sen[3][0] = '\0' ;
    sen[4][0] = '\0' ;
    sen[5][0] = '\0' ;

```

```

textform([self risktxt],sen);
while(sen[i][0]) {
    tmpStr[INDEX] = malloc(strlen(sen[i]) + 1);
    strcpy(tmpStr[INDEX],sen[i++]);
    partoffline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
        GEV_OUTLINE_OFF,GEV_SPACE_OFF);
    Gevrow_add_obj(risk_rows,partoffline);
}
return self;
}

- risk_conditions {
    int i = 0;
    int j = 0;
    int mark;
    int k = 1;
    int condnum;
    int ptline_size = 0;
    int tmpline_size;
    char sent[6][65];

    boxedspace1 = boxed1;
    boxedspace2 = boxed2;
    boxedspace3 = boxed3;
    boxedspace4 = boxed4;
    boxedspace5 = boxed5;

    mark = [self riskresult];
    riskdetails = Gevrow_create(GEV_HORIZONTAL,GEV_OUTLINE_ON,
        GEV_SPACE_OFF) ;
    conditions = Gevrow_create(GEV_VERTICAL,GEV_OUTLINE_OFF,
        GEV_SPACE_ON) ;
    while (strcmp([self riskcondition:i],"don't know") != 0) {
        tmpline_size = strlen([self riskcondition:i++])/55;
        if (tmpline_size > ptline_size)
            ptline_size = tmpline_size;
    }
    ptline_size++;
    condnum = ++i;
    risk_mat = Matrix_create(1,condnum,1,1);
    do {
        condition = Gevrow_create(GEV_VERTICAL,GEV_OUTLINE_ON,
            GEV_SPACE_ON) ;
        sent[0][0] = '\0' ;
        sent[1][0] = '\0' ;
        sent[2][0] = '\0' ;
        sent[3][0] = '\0' ;
        sent[4][0] = '\0' ;
        sent[5][0] = '\0' ;
        i = 0;

        textform([self riskcondition:j],sent);
        while (i < ptline_size) {
            if (sent[i][0] == 0) {
                partoffline = Gevicocar_create(" ",GEV_FONT6,GEV_0,
                    GEV_OUTLINE_OFF,GEV_SPACE_OFF);
                i++;
            }
            else {
                tmpStr[INDEX] = malloc(strlen(sent[i]) + 1);
                strcpy(tmpStr[INDEX],sent[i++]);
                partoffline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,
                    GEV_0,GEV_OUTLINE_OFF,GEV_SPACE_OFF);
            }
            Gevrow_add_obj(condition,partoffline);

```



```

    }
    Gevrow_add_obj(conditions,condition);

    if(k == 1) {
        if (k== mark)
            boxedspace1 = Gevevt_get_botobj(evt);
            Matrix_enter(risk_mat,k, 1,boxedspace1) ;
        }
    else if (k == 2) {
        if (k== mark)
            boxedspace2 = Gevevt_get_botobj(evt);
            Matrix_enter(risk_mat,k, 1,boxedspace2) ;
        }
    else if (k == 3) {
        if (k== mark)
            boxedspace3 = Gevevt_get_botobj(evt);
            Matrix_enter(risk_mat,k, 1,boxedspace3) ;
        }
    else if (k == 4) {
        if (k== mark)
            boxedspace4 = Gevevt_get_botobj(evt);
            Matrix_enter(risk_mat,k, 1,boxedspace4) ;
        }
    else if (k == 5) {
        if (k== mark)
            boxedspace5 = Gevevt_get_botobj(evt);
            Matrix_enter(risk_mat,k, 1,boxedspace5) ;
        }
    }
    k++;
} while(strcmp([self riskcondition:j++], "don't know") != 0) ;

risktoptab = Gevtab_create(voidico, voidmatrix, voidmatrix,
    risk_mat, condnum,1,GEV_OUTLINE_OFF, GEV_SPACE_OFF,
    GEV_FALSE, voidlift, voidlift);

Gevrow_add_obj(riskdetails,Space);
Gevrow_add_obj(riskdetails,conditions);
Gevrow_add_obj(riskdetails,risktoptab);
Gevrow_add_obj(risk_rows,riskdetails);
}

/* ***** */
/*          AMEND METHODS          */
/* ***** */

+ amend_risk_driver {
    id aRisk ;
    int aRiskResult ;

    [self get_num_risk_wdw];
    do {
        Risknum = [Risk getnum] ;
        if (Risknum != 0) {
            aRisk = [riskCltn at:--Risknum];
            [aRisk userIO];
            aRiskResult = [aRisk confirm_risk];
            [aRisk riskresult: aRiskResult];
        }
    } while (Risknum != 0);
    return self;
}

- userIO {
    int res ;
    int i = 0 ;

```

```

int j = 0 ;
int mark ;
char marker ;

risk_rows = Gevrow_create(GEV_VERTICAL, GEV_OUTLINE_ON, GEV_SPACE_OFF);
[self riskamend];
Gevrow_add_obj(risk_rows,quitRiskDriverViewIcon);

    window = Gevwdw_create(risk_rows);
/* window belong graphic screen */
Gevscrgph_add_window(screen, window);

/* positioning and dimensioning in logical reference */
Siz_set(taille, 500, 330);
Pos_set(position, 600, 300);
Gevwdw_set(window, position, taille);

/* display of the window */
Gevscrgph_display_window(screen, window);

return self ;
}

- riskamend {
[self riskdriver_number];
[self risk_title];
[self risk_textual];
[self risk_conditions_amend];
[self risk_help];
}

- risk_conditions_amend {
int i = 0;
int j = 0;
int mark;
int k = 1;
int condnum;
int ptline_size = 0;
int tmp_size;
char sent[6][65];

    boxedspace1 = boxed1;
    boxedspace2 = boxed2;
    boxedspace3 = boxed3;
    boxedspace4 = boxed4;
    boxedspace5 = boxed5;

    mark = [self riskresult];
    riskdetails = Gevrow_create(GEV_HORIZONTAL,GEV_OUTLINE_ON,
                                GEV_SPACE_OFF) ;
    conditions = Gevrow_create(GEV_VERTICAL,GEV_OUTLINE_OFF,
                                GEV_SPACE_ON) ;
    while (strcmp([self riskcondition:i],"don't know") != 0) {
        tmp_size = strlen([self riskcondition:i++])/55;
        if (tmp_size > ptline_size)
            ptline_size = tmp_size;
    }
    ptline_size++;
    condnum = ++i;
    risk_mat = Matrix_create(1,condnum,1,1);
    do {
        condition = Gevrow_create(GEV_VERTICAL,GEV_OUTLINE_ON,
                                    GEV_SPACE_ON) ;
        sent[0][0] = '\0' ;
        sent[1][0] = '\0' ;
        sent[2][0] = '\0' ;

```

```

sent[3][0] = '\0' ;
sent[4][0] = '\0' ;
sent[5][0] = '\0' ;
i = 0;
textform([self riskcondition:j],sent);
while (i < ptline_size) {
    if (sent[i][0] == 0) {
        partoffline = Gevicocar_create(" ",GEV_FONT6,GEV_0,
            GEV_OUTLINE_OFF,GEV_SPACE_OFF);
        i++;
    }
    else {
        tmpStr[INDEX] = malloc(strlen(sent[i]) + 1);
        strcpy(tmpStr[INDEX],sent[i++]);
        partoffline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
            GEV_OUTLINE_OFF,GEV_SPACE_OFF);
    }
    Gevrow_add_obj(condition,partoffline);
}
Gevrow_add_obj(conditions,condition);

if(k == 1) {
    if (k== mark)
        boxedspace1 = Gevevt_get_botobj(evt);
    Matrix_enter(risk_mat,k, 1,boxedspace1) ;
}
else if (k == 2) {
    if (k== mark)
        boxedspace2 = Gevevt_get_botobj(evt);
    Matrix_enter(risk_mat,k, 1,boxedspace2) ;
}
else if (k == 3) {
    if (k== mark)
        boxedspace3 = Gevevt_get_botobj(evt);
    Matrix_enter(risk_mat,k, 1,boxedspace3) ;
}
else if (k == 4) {
    if (k== mark)
        boxedspace4 = Gevevt_get_botobj(evt);
    Matrix_enter(risk_mat,k, 1,boxedspace4) ;
}
else if (k == 5) {
    if (k== mark)
        boxedspace5 = Gevevt_get_botobj(evt);
    Matrix_enter(risk_mat,k, 1,boxedspace5) ;
}
k++;
} while(strcmp([self riskcondition:j++],"don't know") != 0) ;

risktoptab = Gevtab_create(voidico, voidmatrix, voidmatrix,
    risk_mat, condnum,1,GEV_OUTLINE_OFF, GEV_SPACE_OFF,
    GEV_TRUE, voidlift, voidlift);

Gevrow_add_obj(riskdetails,Space);
Gevrow_add_obj(riskdetails,conditions);
Gevrow_add_obj(riskdetails,risktoptab);
Gevrow_add_obj(risk_rows,riskdetails);
}

- (int) confirm_risk {
    int i = 1 ;
    int j;
    int matsize;
    int VALUE = 0;
    BOOL FOUND = NO;

    matsize = 5;

```

```

endofjob = GEV_FALSE ;
while ( endofjob != GEV_TRUE) {
    evt = Gevscrgph_wait_event(screen);
    if ((Gevvt_get_wdw(evt) == Gevwdw_get_wdw(window)) &&
        (Gevvt_get_typ(evt) == GEV_BUTTON)) {

        if (Gevobj_eq(Gevvt_get_botobj(evt),quitRiskDriverViewIcon)) {
            if (VALUE == 0)
                VALUE = [self riskresult];
            Gevscrgph_rem_window(screen,window);
            endofjob = GEV_TRUE;
        }
        else {
            i = 1 ;
            FOUND = NO;
            while (( i <= matsize) && (!FOUND)) {
                if (Gevobj_eq(Gevvt_get_botobj(evt), Matrix_entry(risk_mat,i++,1))) {
                    // deselect values in matrix above and below selected value
                    VALUE = --i;
                    j = VALUE - 1;
                    while ( j > 0 )
                        Gevchr_deselect(Matrix_entry(risk_mat,j--,1));
                    j = VALUE + 1;
                    while ( j <= matsize )
                        Gevchr_deselect(Matrix_entry(risk_mat,j++,1));
                    FOUND = YES;
                }
            }
        }
    }
    Gevchr_deselect(Matrix_entry(risk_mat,VALUE,1));
    i = 0;
    while ( i < INDEX )
        free(tmpStr[i++]);
    INDEX = 0;
    return VALUE;
}

/* ***** */

// Displays the private data for all Riskdriver objects
// in the Risk dataBase
+ list {
    id riskSeq, aRisk ;
    char c ;
    int i = 0 ;

    system("clear") ;
    riskSeq = [riskCltn eachElement] ;
    while(aRisk = [riskSeq next]) {
        [aRisk print] ;
        if(++i % 2 == 0) {
            [LIST_CONTINUE print] ;
            scanf("%c",&c) ;
            system("clear") ;
        }
    } // end list_riskdriver
}

// This method is to ensure that the Rule number entered has a
// corresponding Riskdriver or condition number.
+ validate {
    id aRisk ;
    char *testStr ;
    int risk, condition, cond ;
    int i = 0 ;

```

```

testStr = malloc(8) ;
do { strcpy(testStr,operand[i]) ;
    if (*testStr == 'R') { // operand is a riskdriver condition
        risk = atoi(++testStr) - 1 ;
        while (++testStr != ',') ;
        condition = atoi(++testStr) - 1 ;
        if ([riskCltn size] < risk) {
            VALID = NO ;
            [RISK_SIZE_ERROR print] ;
        }
        else {
            if (condition > 5) {
                VALID = NO ;
                [ERR_RISK_COND print] ;
            }
            else {
                aRisk = [riskCltn at:risk] ;
                if ([aRisk riskcondition:condition] == NULL) {
                    VALID = NO ;
                    [ERR_RISK_COND print] ;
                }
            }
        }
    }
    else {
        cond = atoi(++testStr) ;
        if ([txtCltn size] < cond) {
            VALID = NO ;
            [ERR_COND_NUM print] ;
        }
    } // end if
} while ((strcmp(operator[i++],"THEN") != 0) && VALID) ;
return self;
}

+ windowopen : someobject {
    // positioning and dimensioning logical references
    currentwdw = Gevwdw_create(someobject);
    Gevscrgph_add_window(screen, currentwdw);
    Siz_set(taille,400,300);
    Pos_set(position, 80,50);
    Gevwdw_set(currentwdw,position,taille);
    Gevscrgph_display_window(screen,currentwdw);
    return self;
}

// INSTANCE METHODS

// Changes an instances private data and the risk management area
// asociated with the riskdriver instance.
- maintenance: (int) riskno {
    id risknum ;
    int num, val ;
    int wt, i, j ;

    riskno++ ;
    while (num != 0) {
        [Interface rkmaintenance] ;
        scanf("%d",&num) ;
        switch(num) {
            case 1: // amend Riskdriver title
                printf("\n\n\told title :- %s\n",[self riskdriver]) ;
                [AMEWD_RISKTITLE print] ;
                copycat() ;
                strcpy([self riskdriver],iobuf) ;
                break ;
            case 2: // amend Riskdriver text
                printf("\n\n\told text :- %s\n",[self risktxt]) ;

```

```

        [AMEND_RISKTEXT print] ;
        copycat() ;
        strcpy([self risktxt],iobuf) ;
        break ;
    case 3: // amend Riskdriver conditions
        i = 0 ;
        j = 0 ;
        while (strcmp("don't know",[self riskcondition:i]) != 0)
            printf("\n\n\t%d %s",++j,[self riskcondition:i++]) ;
        [AMEND_CONDITION_NO print] ;
        scanf("%d",&val) ;
        if (val < 1 || val > j)
            [ERR_COND_NUM print] ;
        else {
            [AMEND_CONDITION print] ;
            copycat() ;
            strcpy([self riskcondition:--val],iobuf) ;
        }
        break ;
    case 4: // amend Riskdriver help
        printf("\n\n\told help :- %s\n",[self riskhlp]) ;
        [AMEND_RISKHELP print] ;
        copycat() ;
        strcpy([self riskhlp],iobuf) ;
        break ;
    case 5: // amend Riskdriver weights
        i = 0 ;
        j = 0 ;
        wt = 0 ;
        while ([self riskweight:i] != 0)
            printf("\n\n\t%d %s\t%d",++j,[self riskcondition:i],
                [self riskweight:i++]) ;
        [AMEND_CONDITION_NO print] ;
        scanf("%d",&val) ;
        if (val < 1 || val > j)
            [ERR_COND_NUM print] ;
        else {
            [AMEND_WEIGHT print] ;
            scanf("%d",&wt) ;
            riskweight[--val] = wt ; }
        break ;
    case 6:
        risknum = [Measure create] ;
        [risknum riskarea:riskno] ;
        [Measure addriskAreas: risknum] ;
        break ;
    case 7:
        risknum = [Measure create] ;
        [risknum riskarea:riskno] ;
        [Measure deleteriskAreas: risknum] ;
        [risknum free] ;
        break ;
    default:
        [INVALID_OPTION print] ;
} // end case
} // end while
return self ;
}

```

```

// copy the contents of one Riskdriver onto another
- copy : aRisk {
    int i = 0 ;
    riskdriver = malloc(strlen([aRisk riskdriver])) ;
    strcpy(riskdriver,[aRisk riskdriver]) ;
    risktxt = malloc(strlen([aRisk risktxt])) ;
    strcpy(risktxt,[aRisk risktxt]) ;
    do {

```

```

        riskcondition[i] = malloc(strlen([aRisk riskcondition:i])) ;
        strcpy(riskcondition[i],[aRisk riskcondition:i]) ;
        riskweight[i] = [aRisk riskweight:i] ;
    } while ([aRisk riskweight:i++] != 0) ;
    riskhlp = malloc(strlen([aRisk riskhlp])) ;
    strcpy(riskhlp,[aRisk riskhlp]) ;
    [self riskresult:[aRisk riskresult]] ;
    return self ;
}

// determine the largest weight for a Risk
- (int) largestweight {
    int g = 0 ;
    int maxsize = 0 ;
    int j ;

    while([self riskweight:g] != 0)
    {
        j = [self riskweight:g++] ;
        maxsize = max(j,maxsize);
    }
    return maxsize ;
}

// find the weight of a Risk which is corresponds to the weight result
- (int) selectedweight {
    int i ;

    i = [self riskresult] ;
    return [self riskweight:--i] ;
}

// Transfer the result captured on screen into the object riskresult value.
- scr_value: (int) res {

    if (strcmp(entityName,"Project") == 0)
        [self update_risk:res for:attrName entity:entityName key:"Name"
         instance:[tempprod projectName]];
    if (strcmp(entityName,"Team") == 0)
        [self update_risk:res for:attrName entity:entityName key:"Name"
         instance:[tempprod teamName]];
    if (strcmp(entityName,"Product") == 0)
        [self update_risk:res for:attrName entity:entityName key:"Name"
         instance:[tempprod prodType]];
    if (strcmp(entityName,"Client") == 0)
        [self update_risk:res for:attrName entity:entityName key:"Name"
         instance:[tempprod custName]];

    return self ;
}

- update_risk:(int)newValue for:(STR)anAttrName entity:(STR)anEntityName
  key:(STR)entityKey instance:instanceName {
    char command[256];

    sprintf(command, "UPDATE %s SETMINUS %s = %d WHERE %s = \"%s\"",
              anEntityName, anAttrName,
              newValue, entityKey, [instanceName str]);

    return [IS interpret:command]; }

- (char *) riskdriver { return riskdriver; }

- (int) riskresult {
    if (strcmp(entityName,"Project") == 0)
        return [self retrieve_risk:attrName entity:entityName key:"Name"
         instance:[tempprod projectName]];
    if (strcmp(entityName,"Team") == 0)
        return [self retrieve_risk:attrName entity:entityName key:"Name"

```

```

        instance:[tempprod teamName]];
    if (strcmp(entityName,"Product") == 0)
        return [self retrieve_risk:attrName entity:entityName key:"Name"
            instance:[tempprod prodType]];
    if (strcmp(entityName,"Client") == 0)
        return [self retrieve_risk:attrName entity:entityName key:"Name"
            instance:[tempprod custName]];
}

-(int) retrieve_risk:(STR)anAttrName entity:(STR)anEntityName key:(STR)entityKey
    instance:instanceName {

    id myIS;
    char command[256];

    sprintf(command, "SELECT %s FROM %s WHERE %s = \"%s\"",
        anAttrName, anEntityName, entityKey, [instanceName str]);
    myIS = [IS interpret:command];
    if ([myIS isCorrect])
        return [[[myIS answer] at:0] at:0] asInt];
    else {
        printf("theFlag : %s\n",[myIS theFlag]);
        [IS interpret:"quit"]; }
    }

- riskresult : (int) aRiskResult {
    [self scr_value:aRiskResult] ;
    // riskresult = aRiskResult;
    return self;
}

- (char *) risktxt {return risktxt ; }

- (char *) riskcondition: (int) index {
    return riskcondition[index];
}

- (int) riskweight: (int) index {
    return riskweight[index] ;
}

- (char *) riskhlp { return riskhlp ; }

- risk_help {
}

- entityName: (char *) str{
    entityName = str;
    return self ;
}

- (char *) entityName{
    return entityName ;
}

- attrName:(char *) str {
    attrName = str;
    return self ;
}

- (char *) attrName{
    return attrName ;
}

=:

```



```

//OBJECTIVE-C SOURCE FILE FOR THE CLASS "State";
// THIS CLASS CORRESPONDS TO A STATE IN AN AUTOMATA
#include "objc.h"
#include "wbs.h"

@requires String;
@requires Sequence;
@requires OrdCltn;
@requires Event;
@requires Error;

=State : Object(RiskGroup , Primitive, Collection)

{
// INSTANCE VARIABLES;
id event; // the user event;
id eventCollection; // all the state's events (instances of Event class);
id eventMessageCollection; // the messages to be displayed to the user;
// To each event in eventCollection corresponds
// a message in eventMessageCollection
id error; // the error to display
id errorCollection; // the errors to be displayed (instances of Error class);
id errorMessageCollection; // the error messages to be displayed ;
// To each error in errorCollection corresponds
// a message in errorMessageCollection
id miscMessageCollection; // all other messages to display are grouped in this cltn;
id stateCollection; // all the states to which the state can transit
// to each event in eventCollection corresponds a state
// in stateCollection
id currentObject; // the object on which the user works
id automata ; // the automata in which the state is
id relatedClass; // the class on which the state operates
int typeOfResponse; // type of response is integer or string
int choiceInMenuResponse; // the user response in case of a menu
float floatResponse; // the user response in case of a question-answer
char stringResponse[MAX_SIZE_OF_NODE_NAME]; // the user response in case of a question-answer
int menuops ;
}

// FACTORY METHODS;

// creates a new state and initialise it by default;
// returns the created state
+ new {
    id aState;
    aState = [super new];
    [aState initialise];
    return aState;
}

//INSTANCE METHODS;

// initialises the state by default.
// by default initialises the receiver errorCollection
// and errorMessageCollection with two errors
- initialise {
    [self errorCollection:
        [OrdCltn with:2, [Error new:NO_ERROR],
        [Error new:OUT_OF_MENU_BOUNDS]]];
    [self errorMessageCollection:
        [OrdCltn with:2, NO_ERROR_MESSAGE,
        OUT_OF_MENU_BOUNDS_MESSAGE]];
    return self;
}

```

```
- eventCollection: anEventCollection {
    eventCollection = anEventCollection;
    return self;
}

- eventCollection {
    return eventCollection;
}

- eventMessageCollection: aMessageCollection {
    eventMessageCollection = aMessageCollection;
    return self;
}

- eventMessageCollection {
    return eventMessageCollection;
}

- errorCollection: anErrorCollection {
    errorCollection = anErrorCollection;
    return self;
}

- errorCollection {
    return errorCollection;
}

- errorMessageCollection: anErrorMessageCollection {
    errorMessageCollection = anErrorMessageCollection;
    return self;
}

- errorMessageCollection {
    return errorMessageCollection;
}

- miscMessageCollection: aMiscMessageCollection {
    miscMessageCollection = aMiscMessageCollection;
    return self;
}

- miscMessageCollection {
    return miscMessageCollection;
}

- stateCollection: aStateCollection {
    stateCollection = aStateCollection;
    return self;
}

- stateCollection {
    return stateCollection;
}

-currentObject : aCurrentObject {
```

```

        currentObject = aCurrentObject;
        return self;
    }

    - currentObject {
        return currentObject;
    }

    -automata : anAutomata {
        automata = anAutomata;
        return self;
    }

    - automata {
        return automata;
    }

    - relatedClass : aClass {
        relatedClass = aClass;
        return self;
    }

    - relatedClass {
        return relatedClass;
    }

    - typeOfResponse : (int) aTypeOfResponse {
        typeOfResponse = aTypeOfResponse;
        return self;
    }

    - (int) choiceInMenuResponse {
        return choiceInMenuResponse;
    }

    - (int) menuops {
        return menuops ;
    }

    - (float) floatResponse {
        return floatResponse;
    }

    - (STR) stringResponse {
        return stringResponse;
    }

    // Displays the user messages.
    // Test whether the type of response waited from the user
    // is a choice in a menu or a question response
    - display {
        id aSequence;
        id aMessage;
        int iMax;
        int i;

        iMax = [eventMessageCollection size];

```

```

// 1 - if it is a choice in a menu display the
//      menu items found in eventMessageCollection
if ((iMax >= 1) && (typeOfResponse == CHOICE_IN_MENU)) {
    for (i = 0; i < iMax ; i++)
        printf("%d - %s \n", i+1, [[eventMessageCollection at:i] str]);
    printf("Your choice ?");

}
// 2 - if it a question response displays the
//      only item found in eventMessageCollection
else if ((iMax == 1) &&
        ((typeOfResponse == STRING_RESPONSE) ||
         (typeOfResponse == FLOAT_RESPONSE))) {
    printf("%s ?\n", [[eventMessageCollection at:0] str]);
    printf("?");
}
else
// 3 - if there is no eventMessageCollection
//      generates an error
[self error:[NOT_INITIALISED_STATE_MESSAGE str]];
}

// returns the error which number is anErrorNumber
- findError: (int) anErrorNumber {
    id anError;
    id aSequence;
    aSequence = [errorCollection eachElement];
    while (anError = [aSequence next])
        if ([anError number] == anErrorNumber)
            return anError;
}

// returns the event which number is anEventNumber
- findEvent: (int) anEventNumber {
    id anEvent;
    id aSequence;
    aSequence = [eventCollection eachElement];
    while (anEvent = [aSequence next])
        if ([anEvent number] == anEventNumber)
            return anEvent;
}

// returns the error offset in errorCollection which
// number is anErrorNumber
- (int) findErrorOffset: (int) anErrorNumber {
    id theError;
    theError = [self findError:anErrorNumber];
    return [errorCollection offsetOf: theError];
}

// returns the event offset in eventCollection which
// number is anEventNumber
- (int) findEventOffset: (int) anEventNumber {
    id theEvent;
    theEvent = [self findEvent:anEventNumber];
    return [eventCollection offsetOf: theEvent];
}

// reads the user response and assign the variables
// choiceInMenuResponse or floatResponse or stringResponse
- read {
    switch (typeOfResponse) {
        case CHOICE_IN_MENU :

```

```

        scanf("%d", &choiceInMenuResponse);
        break;
    case FLOAT_RESPONSE :
        scanf("%f", &floatResponse);
        break;
    case STRING_RESPONSE :
        scanf("%s", stringResponse);
        break;
    }
    return self;
}

// test whether the user response in case of a menu
// is inside the menu bounds
- correct {
    return self;
}

// displays the error message which corresponds to the
// instance variable error
- errorMessage {
    int offset;
    offset = [errorCollection offsetOf: error];
    printf("%s \n", [[errorMessageCollection at: offset] str]);
    return self;
}

// treats the user answer.
// in case of a menu, assigns the event variable with the
// one in eventCollection which corresponds to the user menu.
// in case of a question response assigns the event variable
// with the only event in eventCollection
- treat {
    if (typeOfResponse == CHOICE_IN_MENU)
        event = [eventCollection at: (choiceInMenuResponse -1)];
    else if ((typeOfResponse == STRING_RESPONSE) ||
             (typeOfResponse == FLOAT_RESPONSE))
        event = [eventCollection firstElement];
    else;
    return self;
}

// makes the transition to another state if required.
// the following state is the one in stateCollection
// which corresponds to the event
- followingState {
    int offset;
    id aFollowingState;
    if ((stateCollection != nil) &&
        ([event number] != NO_EVENT)) {
        offset = [eventCollection offsetOf: event];
        aFollowingState = [stateCollection at: offset];
        [[aFollowingState automata] currentState: aFollowingState];
        [aFollowingState execute: currentObject];
    }
    return self;
}

```

```
}

```

```
- treatG : (int) eventnumber {
    event = [eventCollection at: eventnumber];
    return self ;
}

// Executes the receiver with a current object.
// While the user response is incorrect :
// displays the user messages, reads the user
// answer, and tests whether the answer is correct,
// then it treats the user answer and insures the
// transition to another state if required.
- execute : aCurrentObject {
    if ([aCurrentObject notEqual: nil])
        [self currentObject: aCurrentObject];
    do {
        error = [self findError: NO_ERROR];
        [self display];
        [self read];
        [self correct];
        if ([error number] != NO_ERROR)
            [self errorMessage];
    }
    while ([error number] != NO_ERROR);
    [self treat];
    [self followingState];
}

=;
```

```
// Objective-C source file for the class Rule
#include <stdio.h>
#include <objc.h>
#include <math.h>
#include "Riskmess.h"
#include "sac_global.h"
```

```
@requires String, OrdCltn, Interface, Txt, Risk, Graphic ;
```

```
extern   Gevscrgph_t   screen;
extern   Gevwdw_t      window , window1, currentwdw;
extern   Gevicocar_t   confirmico,abortico;
extern   Gevicocar_t   quitRuleViewIcon;
extern   Gevlift_t     voidlift;
extern   Gevspa_t      Space ;
extern   Gevtab_t      confirm_tab;
```

```
extern Matrix_t        voidmatrix;
extern Matrix_t        confirm_mat;
extern Siz_t           taille;
extern Pos_t           position;
```

```
extern ruleCltn,riskCltn,txtCltn ;
extern BOOL flag[] ;
```

```
extern Gevrow_t        error_msg_row;
extern Gevmmsg_t       error_msg;
```

```
Gevrow_t               rulenum_row, rule_rows;
Gevicocar_t            rulenumico;
Gevicocar_t            ruleline;
Gevchr_t               rulenum;
```

```
BOOL validate_risk(),validate_cond(), VALID ;
```

```

char *operator[10], *operand[10] ;
char *operators(), *operands() ;
int op1 = 0 ; // operand index
int op2 = 0 ; // operator index
char englishStr[450], ruleStr[80];
int p, pt ;
static int INDEX = 0;
char *tmpStr[20];
Gevent_t evt;
int endofjob;
char *malloc() ;

= Rule : Object (RiskGroup, Collection, Primitive)
{
    char *rules[6] ;
    char condition[3] ;
}

// Initialise operator and operand buffers and enter Rule
+ create {
    VALID = YES ;
    op1 = 0, op2 = 0 ;
    enter_rules() ;
}

// Display Rule format in english and enter associated text
// if saving Rule in ruleCltn
+ add {
    int cnum ;
    char ans ;
    id aRule ;

    cnum = [txtCltn size] ;
    cnum++ ;
    english_txt() ;
    printf("\n\n\n Save Rules Y/N ") ;
    scanf(" %c",&ans) ;
    if (ans == 'Y' || ans == 'y') {
        p = 0 ;
        while (p <= pt) {
            aRule = [Rule add:p:cnum] ;
            [ruleCltn add:aRule] ;
        }
        [txtCltn add:[Txt add]] ;
    }
}

// Rules are added to the rule collection
// counter provides link between Rule and Txt.
+ add:(int)r:(int) counter {
    int k = 0, j = 0 ;
    self = [self new] ;
    do { rules[k] = malloc(6) ;
        strcpy(rules[k++], operand[r]) ;
    } while (*operator[r++] == 'A') ;
    rules[k] = malloc(6) ;
    strcpy(rules[k], "") ;
    riskitoa(counter, condition) ;
    p = r ;
    return self ;
}

+ amend {
    int num ;

    while (num != 0)
    { [Interface ruleamend] ;

```

```

scanf("%d",&num) ;
switch(num) {
    case 1: [Rule amendRule] ;
        break ;
    case 2: [Txt amendText] ;
        break ;
    case 0: break ;
    default: [INVALID_OPTION print] ;
}
} // end while
return self;
}

+ amendRule {
    id tmpRule, aRule ;
    int num ;
    char ans, cont ;

    do { VALID = YES ;
        num = [Rule getnum] ;
        if (num != 0) {
            aRule = [ruleCltn at:--num] ;
            tmpRule = [Rule new] ;
            [tmpRule copy: aRule] ;
            [tmpRule change] ;
            if (VALID) {
                [AMEND_YES print] ;
                scanf("%c",&ans) ;
                if (ans == 'y' || ans == 'Y')
                    [[ruleCltn insert:tmpRule before:aRule] remove:aRule] ;
            }
        }
        [AMEND_CONTINUE print] ;
        scanf("%c",&cont) ;
    } while (cont == 'y' || cont == 'Y') ;
    return self ;
}

// This method is used to delete Rules from the Rulebase
// The deletion of Rules will sometimes cause the deletion
// of the associated Rule txt if no other rule in the RuleBase
// can be associated with a Rule condition.
+ delete {
    id aRule ;
    char ans, cont ;
    int num ;

    do {
        num = [Rule getnum] ;
        if (num != 0) { // num is inside the range of the Rule dataBase
            aRule = [[ruleCltn at:--num] print] ;
            [DELETE_CONFIRM print] ;
            scanf("%c",&ans) ;
            if (ans == 'y' || ans == 'Y')
                [aRule removeRule:num] ;
        }
        [DELETE_CONT print] ;
        scanf("%c",&cont) ;
    } while (cont == 'y' || cont == 'Y') ;
    return self ;
}

+ view {
    id aRule ;
    char cont ;

```



```

int num, condNum ;

do {
    num = [Rule getnum] ;
    if (num != 0) {
        aRule = [[ruleCltn at:--num] print] ;
        condNum = atoi([aRule condition]) ;
        [[txtCltn at:--condNum] print] ;
    }
    [VIEW_CONTINUE print] ;
    scanf("%c",&cont) ;
} while (cont == 'y' || cont == 'Y') ;
return self ;
}

+ view1 {
    id aRule, ruleSeq ;
    char numStr[8] ;
    int i = 0, j = 0 ;
    int num ;

    [self get_num_rule_wdw];
    do {
        num = [Rule getnum] ;
        if (num != 0) {
            riskitoa(num,numStr) ;
            ruleSeq = [ruleCltn eachElement] ;
            while (aRule = [ruleSeq next]) {
                if (j != 0)
                    operator[j-1] = "OR" ;
                if (strcmp([aRule condition],numStr) == 0) {
                    while (strcmp([aRule rules:i],"*") != 0) {
                        operand[j] = [aRule rules:i++] ;
                        operator[j++] = "AND" ;
                    }
                }
            }
            } // end while
        operator[--j] = "THEN" ;
        pt = j ;
        i = 0 ;
        printf("IF ");
        ruleStr[0] = 0;
        strcat(ruleStr,"IF ");
        do {
            strcat(ruleStr,operand[i]);
            strcat(ruleStr," ");
            strcat(ruleStr,operator[i]);
            strcat(ruleStr," ");
            printf("%s ",operand[i]) ;
            printf("%s ",operator[i]) ;
        } while (strcmp(operator[i++],"THEN") != 0) ;
        rule_rows = Gevrow_create( GEV_VERTICAL, GEV_OUTLINE_OFF, GEV_SPACE_OFF);
        [self rule_format];
        [self rule_to_english];
        printf("\n") ;

        Gevchr_deselect(Matrix_entry(confirm_mat, 1, 1));
        Gevchr_deselect(Matrix_entry(confirm_mat, 1, 2));

        Gevrow_add_obj(rule_rows,quitRuleViewIcon);

        window = Gevwdw_create(rule_rows);
        Gevscrph_add_window(screen, window);
        Siz_set(taille,440,290);
        Pos_set(position,250,150);
        Gevwdw_set(window,position,taille);
        Gevscrph_display_window(screen,window);
        [self quit_rule_view];

```

```

    }
    } while (num != 0);
    return self ;
}

+ quit_rule_view {
    int i = 0;

    while (!(Gevobj_eq(Gevvt_get_botobj(evt),quitRuleViewIcon)) ||
            !(Gevobj_eq(Gevvt_get_ttyp(evt),GEV_BUTTON)))
        evt = Gevscrgph_wait_event(screen);
        Gevscrgph_rem_window(screen>window);
        while( i < INDEX )
            free(tmpStr[i++]);
        INDEX = 0;
        Gevchr_deselect(confirmico);
        Gevchr_deselect(abortico);

        return self;
    }

+ list {
    id ruleSeq, aRule ;
    ruleSeq = [ruleCltn eachElement] ;
    while(aRule = [ruleSeq next])
        printf("Rule = %s\n",[aRule print]) ;
    return self;
}

+ update: (int) num {
    id aRule ;
    char *tmpChar, tmpholder[10] ;
    int len, temp ;
    int i, k ;
    double d ;

    // Ignore *tmpChar equal to 'C'
    // only 'R' (Riskdrivers) need updating after deletion.
    i = 0 ;
    len = [ruleCltn size] ;
    while (i < len) { // keep reading the rows
        aRule = [ruleCltn at:i++] ;
        k = 0 ;
        tmpChar = [aRule rules:k] ;
        while (*tmpChar != '*') {
            tmpChar = [aRule rules:k++] ;
            if (*tmpChar == 'R') {
                tmpChar++ ;
                temp = atoi(tmpChar) ;
                if (num < temp) { // decrement rule by one
                    d = atof(tmpChar) ;
                    ftoa(d - 1.0,tmpChar) ;
                }
            }
        } // end of row
    } // no more rows to read
    return self ;
}

+ decrementRuleCondition: (int) val {
    id condSeq, aRule ;
    char *tmpCond ;

```

```

int cond, k ;

condSeq = [ruleCltn eachElement] ;
while (aRule = [condSeq next]) {
    k = 0 ;
    tmpCond = [aRule rules:k] ;
    while (*tmpCond != '*') {
        tmpCond = [aRule rules:k++] ;
        if (*tmpCond == 'C') {
            tmpCond++ ;
            if (atoi(tmpCond) >= val) {
                cond = atoi(tmpCond) - 1 ;
                riskitoa(cond,tmpCond) ;
            }
        }
    }
}
return self ;
}

+ get_num_rule_wdw {

    rulenum_row = Gevrow_create( GEV_VERTICAL, GEV_OUTLINE_OFF, GEV_SPACE_ON);
    rulenumico = Gevicocar_create("Enter Rule Number",GEV_FONT4,
                                GEV_C, GEV_OUTLINE_OFF, GEV_SPACE_ON);

    rulenum = Gevchr_create( " :- ",GEV_FONT4, GEV_FONT3,20,
                            GEV_OUTLINE_OFF, GEV_SPACE_ON);

    error_msg_row = Gevrow_create(GEV_VERTICAL, GEV_OUTLINE_OFF, GEV_SPACE_ON);
    error_msg = Gevmsg_create("",GEV_FONT4,GEV_FONT4,18,GEV_OUTLINE_OFF,GEV_SPACE_ON);
    Gevrow_add_obj(error_msg_row,error_msg);

    //Gevrow_add_obj(risknum_row,top);
    Gevrow_add_obj(rulenum_row,Space);
    Gevrow_add_obj(rulenum_row,rulenumico);
    Gevrow_add_obj(rulenum_row,rulenum);
    Gevrow_add_obj(rulenum_row,Space);

    Gevchr_deselect(confirmico);
    Gevchr_deselect(abortico);

    Gevrow_add_obj(rulenum_row,confirm_tab);
    Gevrow_add_obj(rulenum_row,error_msg_row);

    [self windowopen:rulenum_row];
    return self;
}

+ (char *) enter_rulenum {
    int VALUE;
    char *numchr;

    while (!(Gevobj_eq(Gevvt_get_botobj(evt),rulenum)) ||
            !(Gevobj_eq(Gevvt_get_ttyp(evt),GEV_VALIDATION)))
        evt = Gevscrph_wait_event(screen);
    numchr = Gevchr_get_cont(rulenum);

    while((!(Gevobj_eq(Gevvt_get_botobj(evt),confirmico)) ||
            (!Gevobj_eq(Gevvt_get_botobj(evt),abortico))) &&
            (!Gevobj_eq(Gevvt_get_ttyp(evt),GEV_SELECTED)))
        evt = Gevscrph_wait_event(screen);

    if (Gevobj_eq(Gevvt_get_botobj(evt) ,abortico)) {

```

```

        Gevscrgph_rem_window(screen,currentwdw);
        numchr = ZEROCHAR;
    }
    return numchr;
}

// verifies the rule number entered
+ (int) getnum {
    int num;
    int i;
    char *numStr, tmpStr[10];
    BOOL ERROR_FOUND;

    do {
        ERROR_FOUND = NO;
        numStr = [self enter_rulenum];
        strcpy(tmpStr,numStr);
        i = 0;
        while ((tmpStr[i] != '\0') && !ERROR_FOUND) {
            if (tmpStr[i] < '0' || tmpStr[i] > '9') {
                ERROR_FOUND = YES;
                Gevmsg_set_cont(error_msg,"ERROR : non numeric value entered");
            }
            i++;
        }
        if (!ERROR_FOUND) {
            num = atoi(tmpStr);
            if (num < 0 || num > [ruleCltn size])
                Gevmsg_set_cont(error_msg,"ERROR : invalid rule number");
        }
    } while (num < 0 || num > [ruleCltn size]) ;
    Gevmsg_set_cont(error_msg,"");
    return num;
}

+ (int) confirm_rulenum {
    int VALUE;
    char *numchr;
    endofjob = GEV_FALSE ;
    while ( endofjob != GEV_TRUE) {
        evt = Gevscrgph_wait_event(screen);

        //while(Gevvt_get_ttyp(evt) != GEV_VALIDATION)
        //{
            evt = Gevscrgph_wait_event(screen);
            // printf("evt = %d\n",evt);
            // printf("GEV_VALIDATION = %d\n",GEV_VALIDATION); }
        //(Gevvt_get_ttyp(evt) != GEV_VALIDATION));

        if (Gevvt_get_wdw(evt) == Gevwdw_get_wdw(currentwdw)) {
            if (Gevvt_get_ttyp(evt) == GEV_SELECTED) {
                if (Gevobj_eq(Gevvt_get_botobj(evt), confirmico)) {
                    numchr = Gevchr_get_cont(rulenum);
                    VALUE = 3 ;//atoi(numchr);
                    endofjob = GEV_TRUE ;
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt), abortico)) {
                    Gevscrgph_rem_window(screen,currentwdw);
                    VALUE = 0;
                    endofjob = GEV_TRUE ;
                }
            }
        }
    }
}

```

```

    return VALUE;
}

+ rule_format {
    int i = 0;
    char rulesen[6][65];
    rulesen[0][0] = '\0' ;
    rulesen[1][0] = '\0' ;
    rulesen[2][0] = '\0' ;
    rulesen[3][0] = '\0' ;
    rulesen[4][0] = '\0' ;
    rulesen[5][0] = '\0' ;

    textform(ruleStr,rulesen);
    while(rulesen[i][0]) {
        tmpStr[INDEX] = malloc(strlen(rulesen[i]) + 1);
        strcpy(tmpStr[INDEX],rulesen[i++]);
        ruleline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
                                   GEV_OUTLINE_OFF,GEV_SPACE_OFF);
        Gevrow_add_obj(rule_rows,ruleline);
    }
    return self;
}

+ rule_to_english {
    int i = 0;
    char eng_sen[6][65];
    eng_sen[0][0] = '\0' ;
    eng_sen[1][0] = '\0' ;
    eng_sen[2][0] = '\0' ;
    eng_sen[3][0] = '\0' ;
    eng_sen[4][0] = '\0' ;
    eng_sen[5][0] = '\0' ;

    english_txt();
    textform(englishStr,eng_sen);
    while(eng_sen[i][0]) {
        tmpStr[INDEX] = malloc(strlen(eng_sen[i]) + 1);
        strcpy(tmpStr[INDEX],eng_sen[i++]);
        ruleline = Gevicocar_create(tmpStr[INDEX++],GEV_FONT6,GEV_0,
                                   GEV_OUTLINE_OFF,GEV_SPACE_OFF);
        Gevrow_add_obj(rule_rows,ruleline);
    }
    return self;
}

+ (int) confirm_abort {
    int i = 0;
    int VALUE = 0;
    endofjob = GEV_FALSE ;
    while ( endofjob != GEV_TRUE) {
        evt = Gevscrgph_wait_event(screen);

        if (Gevvt_get_wdw(evt) == Gevwdw_get_wdw(currentwdw)) {
            if (Gevvt_get_typ(evt) == GEV_SELECTED) {
                if (Gevobj_eq(Gevvt_get_botobj(evt), confirmico)) {
                    Gevscrgph_rem_window(screen,currentwdw);
                    VALUE++;
                    while( i < INDEX )
                        free(tmpStr[i++]);
                    INDEX = 0;
                    endofjob = GEV_TRUE ;
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt), abortico)) {
                    Gevscrgph_rem_window(screen,currentwdw);
                    while( i < INDEX )

```

```

        free(tmpStr[i++]);
        INDEX = 0 ;
        VALUE = 0;
        endofjob = GEV_TRUE ;
    }
}
}
return VALUE;
}

+ windowopen : someobject {
    currentwdw = Gevwdw_create(someobject);
    Gevscrgph_add_window(screen, currentwdw);
    Siz_set(taille,400,260);
    Pos_set(position, 50,50);
    Gevwdw_set(currentwdw,position,taille);
    Gevscrgph_display_window(screen,currentwdw);
    return self;
}

- (int) removeRule : (int) num {
    id rule, aRule, ruleSeq ;
    char *tmpCond, *tmpBack, *tmpForward, tmp[3] ;
    int val, forward, back;
    BOOL MATCH = NO ;

    tmpCond = [self condition] ;
    if (num > 0) {
        back = num - 1 ;
        tmpBack = [[ruleCltn at:back] condition] ;
        if (strcmp(tmpCond,tmpBack) == 0)
            MATCH = YES ;
    }
    if ((MATCH == NO) && (num < ([ruleCltn size] - 1))) {
        forward = num + 1 ;
        tmpForward = [[ruleCltn at:forward] condition] ;
        if (strcmp(tmpCond,tmpForward) == 0)
            MATCH = YES ;
    }
    if (MATCH == NO){ // This is the only Rule associated
                      // with a certain condition, therefore
                      // the condition must be deleted and all
                      // the other conditions decremented by one.
        [txtCltn removeAt:atoi(tmpCond) - 1] ;
        ruleSeq = [ruleCltn eachElement] ;
        while (rule = [ruleSeq next]) {
            if ((val = atoi([rule condition])) > atoi(tmpCond)) {
                riskitoa(--val,tmp) ;
                strcpy([rule condition],tmp) ;
            }
        }
        [Rule decrementRuleCondition:atoi(tmpCond)] ;
    }
    [ruleCltn removeAt:num] ;
    return num;
}

- (BOOL) ruleState { // Checks if the instance of the Rule sending
    id tmpRisk ; // the message in the ON state. i.e.

```

```

int j, ci, rd, rc ; // all the rules are true.
BOOL CONTINUE = YES ;
char *tmpRule;

j = 0 ;
tmpRule = [self rules:j] ;
while ((*tmpRule != '*') && CONTINUE) {
    tmpRule = [self rules:j++] ;
    if (*tmpRule == '*')
        ; //empty statement
    else if (*tmpRule == 'R') {
        tmpRule++ ;
        rd = atoi(tmpRule) ;
        rc = (int) (10 * (atoi(tmpRule) - atoi(tmpRule)) + .5) ;
        tmpRisk = [riskCltn at:--rd] ;
        if ([tmpRisk riskresult] != rc) //Riskdriver condition
            CONTINUE = NO ; // not true
    }
    else {
        tmpRule++ ;
        ci = atoi(tmpRule) ;
        if (flag[ci] == NO) //condition text not (switched ON) true
            CONTINUE = NO ;
    }
} // end while along the row
return CONTINUE ;
}

- copy : aRule { // copies the private data of aRule to the instance
    int j = 0 ; // sending thw message, used when a copy of an
                // instance is needed.
    do {
        rules[j] = malloc(6) ;
        strcpy([self rules:j],[aRule rules:j]) ;
    } while (strcmp([aRule rules:j++],"*") != 0) ;
    strcpy ([self condition], [aRule condition]) ;
    return self ;
}

- change {
    char tmp[8], tmpint[8] ;
    int cnum ;
    int i = 0, j = 0 ;

    while(strcmp([self rules:i],"*") != 0)
        printf("\t%d\t%s\n",++j,[self rules:i++]) ;
    printf("\tSelect Option 1 to %d\n",j) ;
    scanf("%d",&cnum) ;
    if (cnum < 1 || cnum > j)
        [COND_NUM_HIGH print] ;
    else {
        [NEW_COMPONENT print] ;
        scanf("%s",tmp) ;
        if (strlen(tmp) < 1) { // empty string to replace
            [self compress:cnum] ; // Rule componemt
            if (strcmp([self rules:0],"*") == 0) {
                VALID = NO ;
                [ERROR_COMPONENT_DELS print] ;
            }
        }
    }
    else { // some text has been assigned to tmp
        operator[0] = "THEN" ;
        operand[0] = tmp ;
        switch (tmp[0]) {
            case 'R': validate_risk(tmp) ;
                break ;
            case 'C': validate_cond(tmp) ;

```

```

        if (VALID)
            [self condition_range:tmp] ;
        break ;
    default: VALID = NO ;
    [ERROR_COMPONENT_TXT print] ;
} // end case
if (VALID) {
    while (tmp[++i] != '\0')
        tmpint[j++] = tmp[i] ;
    if (atoi(tmpint) > [ruleCltn size]) {
        VALID = NO ;
        [RULE_SIZE_ERROR print] ;
    }
}
if (VALID)
    [Risk validate] ;
}
}
if (VALID) { // put amended rule into operand buffer
    printf("Self name %s\n", [self name]) ;
    strcpy([self rules:--cnum], tmp) ;
    [self dump_to_opbuf] ;
    repeated_rules() ;
}
return self ;
}

- compress : (int) k {
    int j ;
    j = k ;
    do {
        strcpy([self rules:k++], [self rules:++j]) ;
    } while(strcmp([self rules:j], "+") != 0) ;
    strcpy([self rules:--k], "") ;
    return self ;
}

- dump_to_opbuf {
    int i = 0 ;

    while(strcmp([self rules:i], "+") != 0) {
        operator[i] = "AND" ;
        operand[i++] = [self rules:i] ;
    }
    pt = --i ;
    strcpy(operator[i], "THEN") ;
    return self ;
}

- condition_range: (char[]) tmp {
    char val[8] ;
    int i = 0 ;
    int j = 0 ;

    while(tmp[i++] != '\0')
        val[j++] = tmp[i] ;
    if (atoi(val) >= atoi([self condition])) {
        VALID = NO ;
        [COND_NUM_HIGH print] ;
    }
    return self ;
}

- printOn: (IOD) anIOD {
    int i = 0 ;

```



```

[super printOn: anIOD] ;
fprintf(anIOD,"%s %s %s %s %s %s %s\n",
        [self rules:i++],
        [self rules:i++],
        [self rules:i++],
        [self rules:i++],
        [self rules:i++],
        [self rules:i++],
        [self condition]) ;
return self ;
}

- (char *) condition {
    return condition ;
}

- (char *) rules: (int) index {
    return rules[index] ;
}

char *operands(strpt)
char *strpt ;
{
    return (operand[op1++] = strpt) ;
}

char *operators(strpt)
char *strpt ;
{
    return (operator[op2++] = strpt) ;
}

int validate_rules(ruleStr,rulesize)    // This module is designed to allow the admin user to
char *ruleStr ; // add new RULES to the Risk Analysis Tool.
int rulesize ; // The adding of new rules must be in a format similar
{
    // to ex.1
    //
    // ex.1 if R1.1 and R2.1 or R3.2 then C1
    //
    // R identifies the Risk driver condition
    // C represents a condition which maybe the part
    // of another rule.
    // Conditions and rules maybe integrated to form rules.
    // ex.2 if R3.2 and C1 then C2

    char testStr[6] ;
    char *strpt ;
    int i = 0, j = 0 ;    // index pointer for rule string
    int ptr = 0 ;

    while (i < rulesize) {
        j = 0 ;
        while(ruleStr[i] == ' ')
            i++ ; // skip spaces
        while(ruleStr[i] != ' ' && ruleStr[i] != '\0')
            testStr[j++] = ruleStr[i++] ;
        testStr[j++] = '\0' ;
        strpt = malloc(sizeof(testStr)) ;
        strcpy(strpt,testStr) ;
        if (ptr++ % 2 == 0)
            operands(strpt) ;
        else
            operators(strpt) ;
    }
    if ( op1 != op2)
        VALID = NO ;
}

```

```

        pt = op1 - 1 ;
        if (VALID)
            validate_operands() ;
        if (VALID)
            validate_operators() ;
    }

validate_operators()
{
    int i = 0 ;

    if (strcmp(operator[pt],"THEN") != 0) {
        VALID = NO ;
        printf("Error : THEN not present at end of Ruleline\n") ;
    }
    while (i <= pt && VALID) {
        switch(operator[i][0]) {
            case 'A': // AND
                if (strcmp(operator[i],"AND") != 0) {
                    VALID = NO ;
                    printf("Error incorrect AND format %s\n",operator[i]);
                    break ;
                }
            case 'O': // OR
                if (strcmp(operator[i],"OR") != 0) {
                    VALID = NO ;
                    printf("Error incorrect OR format %s\n",operator[i]);
                    break ;
                }
            case 'T': // THEN TO be worked on
                if(strcmp(operator[i],"THEN") != 0 && i != pt) {
                    VALID = NO ;
                    printf("Error incorrect THEN format %s\n",operator[i]);
                    break ;
                }
            default:
                VALID = NO ;
                printf("Error : invalid operator %s\n",operator[i]) ;
        }
        i++ ;
    }
}

validate_operands()
{
    int i = 0 ;

    while (i <= pt) {
        switch(operand[i][0]) {
            case 'R': // Risk Driver condition
                validate_risk(operand[i]) ;
                break ;
            case 'C': // condition
                validate_cond(operand[i]) ;
                break ;
            default:
                VALID = NO ;
                printf("Error : invalid operand %s \n",operand[i]) ;
        }
        i++ ;
    }
    if (VALID)
        repeated_rules() ;
}

int repeated_rules()
{
    int ind = 0, i, R1, R2 ;

```

```

while (ind < pt && VALID) {
    i = ind ;
    while (i < pt) {
        if (strcmp2(operand[ind],operand[++i]) == 0) {
            // Risk driver appears twice in the rule statement
            R1 = find_range(ind) ;
            R2 = find_range(i) ;
            if (R1 == R2) {
                printf("Error : risk driver appears twice in rule\n") ;
                VALID = NO ;
            }
        }
        ind++ ;
    }
}

int find_range(i)
int i ;
{
    while (*operator[i++] == 'A') ;
    return i ;
}

enter_rules()
{
    char ruleStr[100];
    int rulesize ;

    printf("Enter new rules in a format similar to below\n\n\n") ;
    printf("\tExample 1 IF R1.1 AND R2.1 OR R4.2 THEN\n\n") ;
    printf("\tExample 2 IF R4.1 AND R6.2 AND C1 OR R5.2 THEN\n\n") ;
    gets(ruleStr) ;
    printf("\tIF ") ;
    gets(ruleStr) ;
    rulesize = strlen(ruleStr) ;
    validate_rules(ruleStr,rulesize) ;
}

BOOL validate_risk(testStr)
char testStr[] ;
{
    int rulesize ;
    int count = 0 ;
    int i = 0 ;
    char temp[6] ;
    id aRisk ;

    rulesize = strlen(testStr) - 1;
    while (i < rulesize && VALID) {
        i = i + 1 ;
        switch(testStr[i]) {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9': break ;
            case '.':
                if (i != (rulesize - 1)) {
                    VALID = NO ;
                }
            }
    }
}

```

```

        printf("Error: incorrect decimal pt position %s",
               testStr) ;
    }
    count++ ;
    if(count > 1) {
        VALID = NO ;
        printf("Error: incorrect decimal points %s",
               testStr) ;
    }
    break ;
default: // invalid risk driver condition format
    VALID = NO ;
    printf("Error: Not a numeric expression %s",testStr) ;
} // end case
} // end while
}

```

```

BOOL validate_cond(testStr)
char testStr[] ;
{
    int rulesize ;
    int i = 0 ;

    rulesize = strlen(testStr) - 1 ;
    while (i < rulesize && VALID)
        switch(testStr[++i]) {
            case '0':
            case '1':
            case '2':
            case '3':
            case '4':
            case '5':
            case '6':
            case '7':
            case '8':
            case '9': break ;
            default: // invalid risk driver condition format
                VALID = NO ;
                printf("Error: invalid condition syntax %s", testStr) ;
        }
}

```

```

english_txt()
{
    // This function is used to convert valid rules entered by the admin
    // user into English.

```

```

    char *tmp ;
    int i = 0 ;
    int y = 0 ;
    englishStr[0] = 0;
    strcat(englishStr,"IF ");
    while ( i <= pt ) {
        tmp = operand[i] ;
        if (*tmp == 'R')
            riskview(tmp) ;
        else
            condview(tmp) ;
        strcat(englishStr," ");
        strcat(englishStr,operator[i]);
        printf(" %s ",operator[i++]) ;
    }
}

```

```

riskview(tmp)

```

```

char *tmp ;
{
    // English representation of the riskdriver located in the riskCltn
    id dummy ;
    char s[2] ;
    int row = 0 ;
    int col = 0 ;

    for (tmp++ ; *tmp >= '0' && *tmp <= '9' ; tmp++)
        row = 10 * row + *tmp - '0' ;
    dummy = [riskCltn at: --row] ;
    if (*tmp == ',') { // digit present display risk condition
        s[0] = *(++tmp) ;
        s[1] = '\0' ;
        col = atoi(s) ;
        printf("%s",[dummy risktxt]) ;
        printf("%s",[dummy riskcondition: col-1]) ;
        strcat(englishStr,[dummy risktxt]);
        strcat(englishStr,[dummy riskcondition: --col]); }
    else
        printf("%s",[dummy risktxt]) ;
}

condview(tmp)
char *tmp ;
{
    // English representation of condition clause located in the txtCltn
    id dummy ;
    int row = 0 ;
    int col = 0 ;

    for (tmp++ ; *tmp >= '0' && *tmp <= '9' ; tmp++)
        row = 10 * row + *tmp - '0' ;
    dummy = [txtCltn at: --row] ;
    strcat(englishStr,[dummy explanation]);
    printf("%s",[dummy explanation]) ;
}
=:

```

```

// Objective-C source file for the class Txt
#include <stdio.h>
#include <objc.h>
#include "libxtrn.h"
#include "Riskmess.h"
@requires String, Interface, OrdCltn ;
extern txtCltn ;
char * malloc();

= Txt : Object (RiskGroup, Collection, Primitive)
{
    char *diagnostic ;
    char *explanation ;
}

+ add { // The admin user enters warning text
    self = [super new] ;
    printf("Enter Advice text\n") ;
    copycat() ;
    diagnostic = malloc(strlen(iobuf)) ;
    strcpy(diagnostic,iobuf) ;
    printf("Enter why text :- ") ;
    copycat() ;
    explanation = malloc(strlen(iobuf)) ;
    strcpy(explanation,iobuf) ;
    return self ;
}

+ amendText {
    id atxt, btxt ;
    int num ;
    char ans, ans1, cont ;

    do {
        [AMEND_TXT_NUM print] ;
        scanf("%d",&num) ;
        if (num < 1 || num > [txtCltn size])
            [ERROR_TXT_NUM print] ;
        else { // valid number to amend text entered
            btxt = [txtCltn at:--num];
            atxt = [Txt new] ;
            [atxt copy:btxt] ;
            [atxt print] ;
            [atxt maintenance] ;
            [AMEND_YES print] ;
            scanf("%c",&ans) ;
            if (ans == 'y' || ans == 'Y')
                [[txtCltn insert:atxt before:btxt] remove:btxt] ;
        }
        [AMEND_CONTINUE print] ;
        scanf("%c",&cont) ;
    } while (cont == 'y' || cont == 'Y') ;
    return self ;
}

- maintenance {
    int num ;

    while (num != 0) {
        [Interface txtmaintenance] ;
        scanf("%d",&num) ;
        switch(num) {
            case 1: // amend text Advice (diagnostic)
                printf("\n\n\t%s",[self diagnostic]) ;
                [AMEND_ADVICE print] ;
                copycat() ;
                diagnostic = malloc(strlen(iobuf)) ;

```

```

        strcpy([self diagnostic],iobuf) ;
        break ;
    case 2:      // amend text why
        printf("\n\n\t%s",[self explanation]) ;
        [AMEWD_WHY print] ;
        copycat() ;
        explanation = malloc(strlen(iobuf)) ;
        strcpy([self explanation],iobuf) ;
        break ;
    case 0 :
        break ;
    default :
        [INVALID_OPTION print] ;
} // end case
} // end while
}

- copy: atxt {
    if ([atxt diagnostic] == NULL)
        diagnostic = malloc(1) ;
    else {
        diagnostic = malloc(strlen([atxt diagnostic])) ;
        strcpy(diagnostic,[atxt diagnostic]) ;
    }
    if ([atxt explanation] == NULL)
        explanation = malloc(1) ;
    else {
        explanation = malloc(strlen([atxt explanation])) ;
        strcpy(explanation,[atxt explanation]) ; }
    return self ;
}

- printOn: (IOD) anIOD {
    [super printOn: anIOD] ;
    fprintf(anIOD,"Diagnostic :- %s\n\n\n Explanation :- %s \n\n\n",
    [self diagnostic],
    [self explanation] ) ;
    return self ;
}

- (char *) diagnostic {
    return diagnostic ;
}

- (char *) explanation {
    return explanation ;
}

- (char *) rptExp {
    printf("\n\nExplanation - %s\n",[self explanation]) ;
}
=;
```

```

// Objective-C source file for the class RiskAutomata
#include "objc.h"
#include "risk.h"
#include "wbs.h"

@requires State;
@requires RiskInitialState;
@requires RiskSecondState;
@requires RiskThirdState ;
@requires RiskFourthState ;
@requires RiskFifthState ;
@requires OrdCltn;

=RiskAutomata : Object(RiskGroup , Primitive, Collection)
{
//INSTANCE VARIABLES;
    id firstState;    // the initial state of the automata;
    id secondState;   // the node operations state of the automata;
    id thirdState ;
    id fourthState ;
    id fifthState ;
    id currentState; // the state of the automata being executed;
    id automata;      // the automata in which the automata is contained
}

//INSTANCE METHODS;

// To initialise the automata:
// - initialises the states
// - define each state
- initialise {
    [self initialiseStates];
    [self defineFirstState];
    [self defineSecondState];
    [self defineThirdState] ;
    [self defineFourthState] ;
    [self defineFifthState] ;
}

// initialises each state
- initialiseStates {
    firstState = [RiskInitialState new];
    [firstState automata:self];
    secondState = [RiskSecondState new];
    [secondState automata:self];
    thirdState = [RiskThirdState new] ;
    [thirdState automata:self] ;
    fourthState = [RiskFourthState new] ;
    [fourthState automata:self] ;
    fifthState = [RiskFifthState new] ;
    [fifthState automata:self] ;
}

// defines the states reachable from the first state
- defineFirstState {
    id anOrdCltn;
    anOrdCltn = [OrdCltn with: 3, secondState,fifthState,UNDEFINED_STATE];
    [firstState stateCollection: anOrdCltn];
}

// defines the states reachable from the second state
- defineSecondState {
    id anOrdCltn;
    anOrdCltn = [OrdCltn with: 3, thirdState,

```



```

                                fourthState,
                                firstState];
    [secondState stateCollection: anOrdCltn];
}

- defineThirdState {
    id anOrdCltn ;
    anOrdCltn = [OrdCltn with: 8, thirdState,
                thirdState, //2
                thirdState, //3
                thirdState, //4
                thirdState, //5
                secondState] ;
    [thirdState stateCollection: anOrdCltn];
}

- defineFourthState {
    id anOrdCltn ;
    anOrdCltn = [OrdCltn with: 6, fourthState,
                fourthState,
                fourthState,
                fourthState,
                fourthState,
                secondState] ;
    [fourthState stateCollection: anOrdCltn] ;
}

- defineFifthState {
    id anOrdCltn ;
    anOrdCltn = [OrdCltn with: 6, fifthState,
                fifthState,
                fifthState,
                fifthState,
                fifthState,
                firstState] ;
    [fifthState stateCollection: anOrdCltn] ;
}

// lunches the automata current state if exist
// else lunches the automata first state
- execute : anObject {
    if (currentState != nil)
        [currentState execute:
         [currentState currentObject]];
    else
        [firstState execute: anObject];
}

// puts aState as the current state of the receiver
// and all the receiver super automata
- currentState: aState {
    currentState = aState;
    if (automata != nil)
        [automata currentState: aState];
    return self;
}

- currentState {
    return currentState;
}

```

```
- currentObject {  
    return [currentState currentObject];  
}  
  
- automata: anAutomata {  
    automata = anAutomata;  
    return self;  
}  
  
- automata {  
    return automata;  
}  
  
- firstState {  
    return firstState;  
}  
  
- secondState {  
    return secondState;  
}  
  
- thirdState {  
    return thirdState ;  
}  
  
- fourthState {  
    return fourthState ;  
}  
  
- fifthState {  
    return fifthState ;  
}  
=;
```

```

#include <ctype.h>
#include <math.h>
#include "objc.h"
#include "risk.h"
#include "wbs.h"
#include "sac_global.h"

@requires String;
@requires OrdCltn;
@requires Event;
@requires RkCltn ;
@requires Graphic;

extern      Gevscrgph_t  screen;
extern      Gevwdw_t     currentwdw;
extern      Gevwdw_t     fifth_wdw;
extern      Gevtab_t     confirm_tab;
extern      Matrix_t     confirm_mat;
extern      Matrix_t     opermat;
extern      BOOL         DELETE_FIFTH_WINDOW ;

Gevicocar_t  operationico,admico,usrico;
Gevtab_t     operationtab;
Gevrow_t     operationwdw;

Gevvt_t      evt;
int          eventnum ;
int          endofjob;

= RiskInitialState : State (RiskGroup , Primitive,Collection)
{
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;

    // inherits its superclass initialisation
    [super initialise];

    // initialise the menu messages
    aMessCltn = [OrdCltn with:3, ADMINISTRATION_MESSAGE,
                USER_MESSAGE,
                QUIT_MESSAGE] ;
    [self eventMessageCollection: aMessCltn] ;

    // initialise the corresponding events
    anEventCltn = [OrdCltn with:3, [Event new:ADMINISTRATION_EVENT],
                        [Event new:USER_EVENT],
                        [Event new:QUIT_EVENT]];
    [self eventCollection: anEventCltn];

    // this is a menu
    [self typeOfResponse:CHOICE_IN_MENU];
}

- display {

    [Graphic operation_wdw];
    if (DELETE_FIFTH_WINDOW)
        Gevscrgph_rem_window(screen,fifth_wdw);

}

```

```

// stay with window until confirmed yes
// if no selection is made open_cal() default is used
- read {
    eventnum = 1 ;
    endofjob = GEV_FALSE;
    while ( endofjob != GEV_TRUE) {
        evt = Gevscrgph_wait_event(screen);

        if (Gevvt_get_wdw(evt) == Gevwdw_get_wdw(currentwdw)) {
            if (Gevvt_get_ttyp(evt) == GEV_SELECTED) {
                if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(opermat,1,1))) {
                    eventnum = 0;
                    Gevchr_deselect(Matrix_entry(opermat,2,1));
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(opermat,2,1))) {
                    eventnum = 1;
                    Gevchr_deselect(Matrix_entry(opermat,1,1));
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(confirm_mat,1,1))) {
                    if (eventnum == 0) {
                        Gevscrgph_rem_window(screen,currentwdw);
                        endofjob = GEV_TRUE;
                    }
                    else if (eventnum == 1) {
                        Gevscrgph_rem_window(screen,currentwdw);
                        endofjob = GEV_TRUE;
                    }
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(confirm_mat,1,2))) {
                    // user is closing down remember to update files
                    eventnum = 2;
                    Gevscrgph_rem_window(screen,currentwdw);
                    endofjob = GEV_TRUE;
                }
            }
        }
    }
}

- treat {
    [super treatG:eventnum];
    switch ([event number]) {

        case ADMINISTRATION_EVENT :
            break ;
        case USER_EVENT :
            break;
        case QUIT_EVENT :
            [RkCltn closeDown] ;    // write data objects onto risk
            exit(0) ;                //and rule collections
            break;
    }
    return self;
}
=:

```

```

// Objective-C source file for the class RiskSecondState
#include "objc.h"
#include "risk.h"
#include "wbs.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence ;
@requires Interface ;

extern BOOL VALID ;
extern id riskCltn ;

=RiskSecondState : State(RiskGroup , Primitive, Collection)
{
//INSTANCE VARIABLES;
}

//INSTANCE METHODS;

- initialise {
    id anEventCltn;
    id aMessCltn;

    // inherits its superclass initialisation
    [super initialise];

    // initialise the menu messages
    aMessCltn = [OrdCltn with:3, EXIT_MESSAGE,
                  RULE_MESSAGE,
                  RISK_MESSAGE] ;
    [self eventMessageCollection: aMessCltn] ;

    // initialise the events
    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:RULE_EVENT]]; //1
    [anEventCltn add: [Event new:RISKDRIVER_EVENT]]; //2
    [anEventCltn add: [Event new:EXIT_EVENT]]; //0
    [self eventCollection: anEventCltn];
    // it is a menu
    [self typeOfResponse:CHOICE_IN_MENU];
}

- display {
    menuops = [Interface adminscr] ;
    return self ;
}

// Calls the treatment corresponding to the event

- treat {

    [super treat];

    switch ([event number]) {

    case RISKDRIVER_EVENT :
        // [self riskdriver_op];
        break;

    case RULE_EVENT :
        // [self rule_op];
        break;
    }
}

```

```
        case EXIT_EVENT:
            // exit(0);
            break;
    }
    return self;
};
```

```

// Objective-C source file for the class RiskThirdState
#include "objc.h"
#include "risk.h"
#include "wbs.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence;
@requires Risk ;
@requires Interface ;

extern riskCltn;

= RiskThirdState : State(RiskGroup , Primitive, Collection)
{
//INSTANCE VARIABLES;
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;

    // inherits its superclass initialisation
    [super initialise];
    aMessCltn = [OrdCltn with:6, EXIT_MESSAGE,
        ADD_RD_MESSAGE,
        DELETE_RD_MESSAGE,
        VIEW_RD_MESSAGE,
        AMEND_RD_MESSAGE,
        LIST_RD_MESSAGE] ;
    [self eventMessageCollection: aMessCltn] ;

    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:ADD_EVENT]];
    [anEventCltn add: [Event new:DELETE_EVENT]];
    [anEventCltn add: [Event new:VIEW_EVENT]];
    [anEventCltn add: [Event new:AMEND_EVENT]];
    [anEventCltn add: [Event new:LIST_EVENT]] ;
    [anEventCltn add: [Event new:EXIT_EVENT]];
    [self eventCollection: anEventCltn];

    [self typeOfResponse:CHOICE_IN_MENU];
}

- add {
    id aRisk ;
    aRisk = [Risk create] ;
    [riskCltn add:aRisk] ;
    return self ;
}

- delete {
    [Risk delete] ;
    return self ;
}

- amend {
    [Risk amend] ;
    return self ;
}

```

```
- view {
    [Risk view] ;
    return self ;
}

- list {
    [Risk list] ;
    return self ;
}

- display {
    menuops = [Interface riskscr] ;
    return self ;
}

// Calls the treatment corresponding to the event

- treat {

    [super treat];

    switch ([event number]) {
    case ADD_EVENT :
        [self add];
        break;
    case DELETE_EVENT :
        [self delete];
        break;
    case VIEW_EVENT :
        [self view];
        break;
    case AMEND_EVENT :
        [self amend];
        break;
    case LIST_EVENT :
        [self list];
        break;
    case EXIT_EVENT:
        //      exit(0);
        break;
    }
    return self;
}

=;
```



```

// Objective-C source file for the class RiskFourthState
#include "objc.h"
#include "risk.h"
#include "wbs.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence;
@requires Rule ;
@requires Txt ;
@requires Risk ;
@requires Interface ;

extern ruleCltn, txtCltn ;
extern VALID ;

= RiskFourthState : State(RiskGroup , Primitive, Collection)

{
//INSTANCE VARIABLES;
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;

    // inherits its superclass initialisation
    [super initialise];

    aMessCltn = [OrdCltn with:6 ,EXIT_MESSAGE,
                  ADD_MESSAGE,
                  DELETE_MESSAGE,
                  VIEW_MESSAGE,
                  AMEND_MESSAGE,
                  LIST_MESSAGE] ;
    [self eventMessageCollection: aMessCltn] ;

    // initialise the events

    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:ADD_EVENT]];
    [anEventCltn add: [Event new:DELETE_EVENT]];
    [anEventCltn add: [Event new:VIEW_EVENT]];
    [anEventCltn add: [Event new:AMEND_EVENT]];
    [anEventCltn add: [Event new:LIST_EVENT]] ;
    [anEventCltn add: [Event new:EXIT_EVENT]];
    [self eventCollection: anEventCltn];

    [self typeOfResponse:CHOICE_IN_MENU];
}

- add {
    [Rule create] ;
    if (VALID) // Check that Rules entered correspond
               [Risk validate] ; // to a riskdriver condition
    if (VALID) // If Rules are valid enter to RuleBase
               [Rule add] ;
    return self ;
}

- delete {

```

```

    [Rule delete] ;
    return self ;
}

- amend {
    [Rule amend] ;
    return self ;
}

- view {
    [Rule view] ;
    return self ;
}

- list {
    [Rule list] ;
    return self ;
}

- display {
    menuops = [Interface rulescr] ;
    return self ;
}

// Calls the treatment corresponding to the event
- treat {
    [super treat];

    switch ([event number]) {
    case ADD_EVENT :
        [self add];
        break;
    case DELETE_EVENT :
        [self delete];
        break;
    case VIEW_EVENT :
        [self view];
        break;
    case AMEND_EVENT :
        [self amend];
        break;
    case LIST_EVENT :
        [self list];
        break;
    case EXIT_EVENT :
        break;
    }
    return self;
}
=:

```

```

// Objective-C source file for the class RiskFifthState
#include <math.h>
#include "objc.h"
#include "risk.h"
#include "wbs.h"
#include "sac_global.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence;
@requires Graphic ;
@requires Risk ;
@requires Rule ;
@requires User ;

extern      BOOL DELETE_FIFTH_WINDOW ;
extern      BOOL VALID ;
extern      id riskCltn ;
extern      Gevscrgph_t   screen;
extern      Gevwdw_t      currentwdw;
extern      Matrix_t      userops_mat;
extern      Gevicocar_t   confirmico, abortico;

Gevwdw_t     result_wdws[10];
Gevwdw_t     aResultWdw;
Gevwdw_t     fifth_wdw;
Gevvt_t      evt;

int endofjob ;
int eventnum ;
int res_wdw_idx = 0;

=RiskFifthState : State(RiskGroup , Primitive, Collection)
{
// no INSTANCE VARIABLES
}

- initialise {
    id anEventCltn;
    id aMessCltn;

    // inherits its superclass initialisation
    [super initialise];

    aMessCltn = [OrdCltn with:6, EXIT_MESSAGE,
                    USER_RISKDRIVER_MESSAGE,
                    USER_RULE_MESSAGE,
                    USER_AMEND_RISKDRIVER_MESSAGE,
                    USER_AMEND_ALL_RISKDRIVER_MESSAGE,
                    USER_RESULTS_MESSAGE] ;
    [self eventMessageCollection: aMessCltn] ;

    // initialise the events
    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:USER_RISKDRIVER_EVENT]];
    [anEventCltn add: [Event new:USER_RULE_EVENT]];
    [anEventCltn add: [Event new:USER_AMEND_RISKDRIVER_EVENT]];
    [anEventCltn add: [Event new:USER_AMEND_ALL_RISKDRIVER_EVENT]];
    [anEventCltn add: [Event new:USER_RESULTS]];
    [anEventCltn add: [Event new:EXIT_EVENT]];
    [self eventCollection: anEventCltn];

    [self typeOfResponse:CHOICE_IN_MENU];
}

```

```

- riskdriver_view {
    [Risk view] ;
    return self ;
}

- rule_view {
    [Rule view1] ;
    return self ;
}

- amend_riskdriver {
    [Risk amend_risk_driver];
    return self ;
}

- amend_all_riskdriver {
    [User allUserIO] ;
    return self ;
}

- results {
    [User results] ;
    return self ;
}

- display {
    int k;
    [Graphic userops_wdw];
    k = 0;

    while (k < res_wdw_idx)
        Gevscrph_rem_window(screen,result_wdes[k++]);
    res_wdw_idx = 0;
    return self ;
}

- read {
    int i = 1 ;
    int j;
    int VALUE = 0;
    BOOL FOUND = NO;

    endofjob = GEV_FALSE ;
    while ( endofjob != GEV_TRUE) {
        evt = Gevscrph_wait_event(screen);
        if (Gevvt_get_wdw(evt) == Gevwdw_get_wdw(currentwdw)) {
            if (Gevvt_get_ttyp(evt) == GEV_SELECTED) {
                if (Gevobj_eq(Gevvt_get_botobj(evt), confirmico)) {
                    if (VALUE == 0) // confirm without selection
                        eventnum = 0; // ico one is used as default
                    Gevscrph_rem_window(screen,currentwdw);
                    endofjob = GEV_TRUE ;
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt), abortico)) {
                    eventnum = 5 ;
                    fifth_wdw = currentwdw;
                    DELETE_FIFTH_WINDOW = YES;
                    endofjob = GEV_TRUE ;
                }
            }
            else {
                i = 1 ;
                FOUND = NO;
                while (( i <= 5) && (!FOUND)) {
                    if (Gevobj_eq(Gevvt_get_botobj(evt), Matrix_entry(userops_mat,i++,1))) {
                        VALUE = --i;
                        j = VALUE - 1;

```

```

        while ( j > 0 )
            Gevchx_deselect(Matrix_entry(userops_mat,j--,1));
        j = VALUE + 1;
        while ( j <= 5 )
            Gevchx_deselect(Matrix_entry(userops_mat,j++,1));
        FOUND = YES;
        eventnum = --VALUE;
    }
}
}
}
}

- treat {
    [super treatG:eventnum];

    switch ([event number]) {

    case USER_RISKDRIVER_EVENT :
        [self riskdriver_view];
        break;

    case USER_RULE_EVENT :
        [self rule_view];
        break;

    case USER_AMEND_RISKDRIVER_EVENT :
        [self amend_riskdriver] ;
        break ;

    case USER_AMEND_ALL_RISKDRIVER_EVENT :
        [self amend_all_riskdriver] ;
        break ;

    case USER_RESULTS :
        [self results] ;
        break ;

    case EXIT_EVENT:
        break;
    }
    return self;
}

=:

```

Appendix E

Calendar Tool Classes

```

#include <stdio.h>
#include <objc.h>
#include <math.h>
#include "sac_global.h"
#include "main.h"
#include "wbs.h"
#include "cal.h"

extern int errno;
extern FILE *yyin;

id anEventDataCltn;
id aCALLoad;

@requires CalAutomata;
@requires CALLoad;
@requires IS;
@requires String;
@requires OrdCltn ;
@requires Graphic;
id anAutomata;
static STR savingFileName = "cal.io";

```

```

Pos_t      position;
Siz_t      taille;
Rpos_t     rposition;
Rsiz_t     rtaille;
Matrix_t   risktab ;
Gevlift_t  lifthor, liftver ;
Gevtab_t   caltoptab;
Gevwdw_t   currentwdw;

Gevscrgph_t  screen;
Gevicocar_t  calhelpico ;
Gevicocar_t  notitle ;
Gevicocar_t  titleico;
Gevicocar_t  bottom, notitle ;
Gevicocar_t  voidico;

// tables to hold matrix
Gevtab_t     confirmatab,conftoptab;
Gevtab_t     risktoptab;

Gevrow_t     someobject ;
Gevrow_t     conditions, condition;
Gevrow_t     top, topright ;

Gevwdw_t     testwdw ;
Gevscrgph_t  screen;
Gevspa_t     Space, boxedspace;

Gevrow_t     error_msg_row;
Gevmsg_t     error_msg;
Gevvt_t      evt;

Gevlift_t    voidlift;
Rpos_t       p1, p2, p3;
Lls_t        couples;
Siz_t        taille;
Pos_t        position;

Matrix_t     voidmatrix;

int           endofjob;

```

```

=(CalGroup, Primitive,Collection)

main( argc , argv)
    int argc;
    char * argv[];
{
    extern BOOL msgFlag;
    id base;
    id myIs;

    if (*argv[1] == 't')
        msgFlag = YES;

    if (strcmp(argv[2], "") != 0) {
        base = [String str:"database "];
        [base concatSTR:argv[2]];
    }
    else {
        printf("You must enter the database name\n");
        [IS interpret:"quit"];
    }

    create_fifo();

    [IS interpret:[base str]];

    aCALLoad = [CALLoad new];
    [aCALLoad loadDM];

    [Graphic create];
    anAutomata = [CalAutomata readFrom: savingFileName];

    if (anAutomata == nil){
        anAutomata = [CalAutomata new];
        [anAutomata initialise];
        [anAutomata execute: nil];
    }
    else
        // lunch the saved automata state
        [anAutomata execute: nil];
}

/*****
create_fifo()
{
    char *self_fifo = "/tmp/self_fifoXXXXXX",*mktemp();
    int d;

    mkknod((self_fifo = mktemp(self_fifo)), 0010600, 0);
    d = open(self_fifo,O_RDWR|O_NDELAY,0);

    close(d);
    yyin = fopen(self_fifo,"r+");
}

save()
{
    // save the automata state on file savingFileName
    [anAutomata storeOn: savingFileName];
}

@classes(AsciiFiler)
@messages()

```



```

#include <objc.h>
#include <ctype.h>
#include "date.h"

extern long datescale ;
extern id dateCltn ;
extern long longdate;
extern id dateStrCltn ;
extern id aCALLoad ;

@requires String ;
@requires OrdCltn ;
@requires IS ;

= Date : Object (CalGroup, Primitive)
{
    long day_number; // days since Jan 1st 1970
    BOOL day_imports ;
    BOOL day_products ;
    BOOL day_meetings ;
    BOOL day_personnel ;
    BOOL day_work_in_progress ;
    id anImportCltn;
    id aProductCltn;
    id aProgressCltn;
    id aMeetingsCltn;
    id aPersonnelCltn;
}

+ dummy {
    id tmp;
    tmp = [self new];
    return tmp;
}

//Create a new date by using the superclass 'new' method
+ create { // Default to create instance of date which equals system date
    id newdate;
    gettimeofday(&timesignal,&timediff) ;
    newdate = [self new];
    [newdate day_number:timesignal.tv_sec] ;
    [newdate day_imports:NO] ;
    [newdate day_products:NO] ;
    [newdate day_meetings:NO] ;
    [newdate day_personnel:NO] ;
    [newdate day_work_in_progress:NO] ;
    return newdate;
}

// allows the use to specify the date instance
+ create:(long)mart2 {
    id newdate ;
    newdate = [self new] ;
    [newdate day_number:mart2] ;
    [newdate day_imports:NO] ;
    [newdate day_products:NO] ;
    [newdate day_meetings:NO] ;
    [newdate day_personnel:NO] ;
    [newdate day_work_in_progress:NO] ;
    return newdate ;
}

+ (STR)plusDate:(long)aScale {
    id aNewDate;
    char aStrDate[14];

    aNewDate = [self new];
    [aNewDate day_number:aScale];
}

```

```

    strcpy(aStrDate, [aNewDate convert_to_digit]);

    return aStrDate;
}

- (STR)convert_to_digit {
    char aStrDate[14];

    strcpy(aStrDate, [[String stringWithFormat:@"%d", [self year]] str]);
    strcat(aStrDate, "-");
    if ([self month] < 10)
        strcat(aStrDate, "0");
    strcat(aStrDate, [[String stringWithFormat:@"%d", [self month]] str]);
    strcat(aStrDate, "-");
    if ([self dayinMonth] < 10)
        strcat(aStrDate, "0");
    strcat(aStrDate, [[String stringWithFormat:@"%d", [self dayinMonth]] str]);

    return aStrDate;
}

+ forward {
    id date ;
    long lastdate ;

    date = [dateCltn lastElement] ;
    lastdate = [date day_number] ;
    lastdate += datescale ;
    [dateCltn removeFirst] ;
    [dateCltn add:[Date create:lastdate]] ;
    return self ;
}

+ back {
    id date ;
    long firstdate ;

    date = [dateCltn firstElement] ;
    firstdate = [date day_number] ;
    firstdate -= datescale ;
    [dateCltn removeLast] ;
    [dateCltn addFirst:[Date create:firstdate]] ;
    return self ;
}

+ (int) establish_month : (char *) dateStr {
    char * subDate ;
    char tmpStr[14] ;
    int datelen, i = 0 , j = 0 ;
    BOOL FOUND = NO ;

    subDate = dateStr ;
    datelen = strlen(dateStr) ;
    while(!isalpha(*subDate)) && i++ < datelen
        subDate++ ;
    if (i > datelen)
        printf("Error month str not found\n") ;
    else {
        while(isalpha(*subDate))
            tmpStr[j++] = *subDate++ ;
        tmpStr[j] = '\0' ;
    }
    i = 0 ;
    while (!FOUND && i < 12) {
        if (strcmp2(tmpStr, month_name[i++]) == 0)
            FOUND = YES ;
    }
}

```

```

    if(FOUND)
        return --i ;
    else
        return 13 ;
}

+ (int) establish_day :(char *) dateStr {
    char * subDate ;
    char tmpStr[14] ;
    int datelen, i = 0 , j = 0 ;
    BOOL FOUND = NO ;

    subDate = dateStr ;
    datelen = strlen(dateStr) ;
    while((!isdigit(*subDate)) && i++ < datelen)
        subDate++ ;
    if (i > datelen)
        printf("Error day nimber not found\n") ;
    else {
        while(isdigit(*subDate) && i++ < datelen)
            tmpStr[j++] = *subDate++ ;
        tmpStr[j] = '\0' ;
    }
    return atoi(tmpStr) ;
}

+ (int) establish_year :(char *) dateStr {
    char * subDate ;
    char tmpStr[14] ;
    int datelen, i = 0 , j = 0 ;
    BOOL FOUND = NO ;

    subDate = dateStr ;
    datelen = strlen(dateStr) ;
    while((!isdigit(*subDate)) && i++ < datelen)
        subDate++ ;
    if (i > datelen)
        printf("Error day nimber not found\n") ;
    else {
        while(isdigit(*subDate) && i++ < datelen)
            subDate++ ;
        if (i > datelen)
            printf("Error day nimber not found\n") ;
        else {
            while((!isdigit(*subDate)) && i++ < datelen)
                subDate++ ;
            if (i > datelen)
                printf("Error day nimber not found\n") ;
            else {
                while(isdigit(*subDate) && i++ < datelen)
                    tmpStr[j++] = *subDate++ ;
                tmpStr[j] = '\0' ;
            }
        }
    }
    return atoi(tmpStr) ;
}

+ getdates {
    int i = 0;
    id dateSeq ;
    id aDate;
    id date;
    id dateStr;

    dateStrCltn = [OrdCltn new];
    while (i++ < 31) {

```

```

        aDate = [Date create:longdate];
        [dateCltn add:[Date create:longdate]] ;
        longdate += datescale ;
    }
    dateSeq = [dateCltn eachElement];
    while (date = [dateSeq next]) {
        dateStr = [String new:13];
        sprintf(dateStr, " %d%c%d%c%d ",[date year],'-',
            [date month],'-',[date dayinMonth]);
        [dateStrCltn add:dateStr];
    }
    return self;
}

- (char *) convert_date_to_str {
    int day, mon, year ;
    char dateStr[12], monStr[3], dayStr[3], yearStr[5] ;

    dateStr[0] = '\0' ;
    mon = [self month] ;
    day = [self dayinMonth] ;
    year = [self year] ;

    if (mon < 10)
        strcat(dateStr,"0") ;
    itoa(mon,monStr) ;
    strcat(dateStr,monStr) ;
    strcat(dateStr,"/") ;
    if (day < 10)
        strcat(dateStr,"0") ;
    itoa(day,dayStr) ;
    strcat(dateStr,dayStr) ;
    strcat(dateStr,"/") ;
    itoa(year,yearStr) ;
    strcat(dateStr,yearStr) ;
    return dateStr ;
}

- day_number:(long)day1 {
    day_number = day1;
    return self;
}

- (BOOL) day_imports : (BOOL) bool_val {
    char aStrDate[12], aNewStrDate[12];
    id aCltn;
    int i,j;

    anImportCltn = [OrdCltn new];

    strcpy(aStrDate, [self convert_to_digit]);
    strcpy(aNewStrDate, [Date plusDate:[self day_number]+ datescale]);

    aCltn = [aCALLoad taskCltn];

    for (i=0; i<[aCltn size]; i++) {
        if ((strcmp(aStrDate,[[aCltn at:i] startDate] str)) <= 0) &&
            (strcmp(aNewStrDate,[[aCltn at:i] startDate] str)) > 0) {
            for (j=0; j <[[aCltn at:i] consumedCltn] size]; j++)
                [anImportCltn add:[[aCltn at:i] consumedCltn] at:j]];
        }
    }
    if ([anImportCltn size] > 0)
        day_imports = YES ;
    else
        day_imports = NO ;
    return day_imports ;
}

```

```

}

- (BOOL) day_products : (BOOL) bool_val {
    char aStrDate[12], aNewStrDate[12];
    id aCltn;
    int i;

    aProductCltn = [OrdCltn new];

    strcpy(aNewStrDate, [Date plusDate:[self day_number]+ datescale]);
    strcpy(aStrDate, [self convert_to_digit]);

    aCltn = [aCALLoad taskCltn];

    for (i=0; i<[aCltn size]; i++) {
        if ((strcmp(aStrDate,[[aCltn at:i] endDate] str]) <= 0) &&
            (strcmp(aNewStrDate,[[aCltn at:i] endDate] str]) > 0)) {
            [aProductCltn add:[aCltn at:i] product]];
        }
    }

    if ([aProductCltn size] > 0)
        day_products = YES ;
    else
        day_products = NO ;
    return day_products ;
}

- (BOOL) day_meetings : (BOOL) bool_val {
    day_meetings = bool_val ;
    return day_meetings ;
}

- (BOOL) day_personnel : (BOOL) bool_val {
    char aStrDate[12], aNewStrDate[12];
    id aCltn1;
    id aCltn;
    id aWork;
    int i,j;

    aCltn1 = [OrdCltn new] ;

    strcpy(aNewStrDate, [Date plusDate:[self day_number]+ datescale]);
    strcpy(aStrDate, [self convert_to_digit]);

    aCltn = [aCALLoad taskCltn];
    for (i=0; i<[aCltn size]; i++)
        for (j=0; j<[[aCltn at:i] workCltn] size]; j++) {
            aWork = [[aCltn at:i] workCltn] at:j];
            if ((strcmp(aStrDate,[[aWork toDate] str]) < 0) &&
                (strcmp(aNewStrDate,[[aWork toDate] str]) >= 0))
                [aCltn1 addIfAbsentMatching:[aWork resourceName]];
        }
    for (i=0; i<[aCltn size]; i++)
        for (j=0; j<[[aCltn at:i] workCltn] size]; j++) {
            aWork = [[aCltn at:i] workCltn] at:j];
            if ((strcmp(aStrDate,[[aWork fromDate] str]) <= 0) &&
                (strcmp(aNewStrDate,[[aWork fromDate] str]) > 0))
                [aCltn1 addIfAbsentMatching:[aWork resourceName]];
        }
    for (i=0; i<[aCltn size]; i++)
        for (j=0; j<[[aCltn at:i] workCltn] size]; j++) {
            aWork = [[aCltn at:i] workCltn] at:j];
            if ((strcmp(aStrDate,[[aWork fromDate] str]) > 0) &&
                (strcmp(aNewStrDate,[[aWork toDate] str]) < 0))
                [aCltn1 addIfAbsentMatching:[aWork resourceName]];
        }
}

```

```

    if ([aCltn1 size] > 0) {
        aPersonnelCltn = aCltn1;
        day_personnel = YES ;
    } else
        day_personnel = NO ;

    return day_personnel ;
}

- (BOOL) day_work_in_progress : (BOOL) bool_val {
    char aStrDate[12], aNewStrDate[12];
    id aCltn1;
    id aCltn;
    id aTask;
    int i;

    aCltn1 = [OrdCltn new] ;

    strcpy(aNewStrDate, [Date plusDate:[self day_number]+ datescale]);
    strcpy(aStrDate, [self convert_to_digit]);

    aCltn = [aCALLoad taskCltn];
    for (i=0; i<[aCltn size]; i++) {
        aTask = [aCltn at:i];
        if ((strcmp(aStrDate,[[aTask endDate] str]) < 0) &&
            (strcmp(aNewStrDate,[[aTask endDate] str]) >= 0))
            [aCltn1 addIfAbsentMatching:[aTask taskName]];
    }

    for (i=0; i<[aCltn size]; i++) {
        aTask = [aCltn at:i];
        if ((strcmp(aStrDate,[[aTask startDate] str]) <= 0) &&
            (strcmp(aNewStrDate,[[aTask startDate] str]) > 0))
            [aCltn1 addIfAbsentMatching:[aTask taskName]];
    }

    for (i=0; i<[aCltn size]; i++) {
        aTask = [aCltn at:i];
        if ((strcmp(aStrDate,[[aTask startDate] str]) > 0) &&
            (strcmp(aNewStrDate,[[aTask endDate] str]) < 0))
            [aCltn1 addIfAbsentMatching:[aTask taskName]];
    }

    if ([aCltn1 size] > 0) {
        day_work_in_progress = YES ;
        aProgressCltn = aCltn1;
    } else
        day_work_in_progress = NO ;

    return day_work_in_progress ;
}

- (BOOL)exist:aName in:aCltn {
    int i;
    BOOL aBool;

    aBool = NO;
    for (i=0; i< [aCltn size]; i++)
        if (strcmp([aName str],[aCltn at:i] str) == 0)
            aBool = YES;
    return aBool;
}

- (long)day_number { return day_number; }

```

```

- (BOOL) day_imports { return day_imports ; }

- (BOOL) day_products { return day_products ; }

- (BOOL) day_meetings { return day_meetings ; }

- (BOOL) day_personnel { return day_personnel ; }

- (BOOL) day_work_in_progress { return day_work_in_progress ; }

- correspondingCltnTo:aLine {
switch (aLine) {
case 1 :
    return anImportCltn;
    break;
case 2 :
    return aProductCltn;
    break;
case 3 :
    return aPersonnelCltn;
    break;
case 4 :
    return aMeetingsCltn;
    break;
case 5 :
    return aProgressCltn;
    break;
}
}

- (long) subday {
return self->day_number - 86400;
}

// compare instance of Date
- (int) compare:bDate {
if(self->day_number > bDate->day_number)
    return(1) ;
else if(self->day_number == bDate->day_number)
    return(0) ;
else
    return(-1) ;
}

- (int)LT: bDate { // Less Than bDate
if([self compare:bDate] == -1)
    return(1) ;
else
    return(0) ;
}

- (int)LE: bDate { // Less than or Equal to bDate
if([self compare:bDate] != 1)
    return(1) ;
else
    return(0) ;
}

- (int)EQ: bDate { // Equals bDate
if([self compare:bDate] == 0)
    return(1) ;
else
    return(0) ;
}

- (int)GE: bDate { // Greater than or Equals bDate
if([self compare:bDate] != -1)

```

```

        return(1) ;
    else
        return(0) ;
}

- (int)GT: bDate { // Greater Than bDate
    if([self compare:bDate] == 1)
        return(1) ;
    else
        return(0) ;
}

// convert the calendar date (i.e. 1989-02-16) into a long
+ (long) convert_date_to_long : (STR) aDateStr {
    char tmp[5];
    int j = 0;
    int i = 0;
    int day, month, year;
    int leapyears;
    long dateasLong ;

    while (aDateStr[i] != '-')
        tmp[j++] = aDateStr[i++];
    tmp[j] = '\0';
    year = atoi(tmp);

    i++;
    j = 0;
    while (aDateStr[i] != '-')
        tmp[j++] = aDateStr[i++];
    tmp[j] = '\0';
    month = atoi(tmp);

    i++;
    j = 0;
    while (aDateStr[i])
        tmp[j++] = aDateStr[i++];
    tmp[j] = '\0';
    day = atoi(tmp);
    leapyears = (year - 1968) / 4;
    year -= 1970;

    dateasLong = ((year * 31536000) + (month * 86400) + (--day * 86400) + (leapyears * 86400)) ;

    return dateasLong;
}

// converts a long into a date which has yyyy-mm-dd format
+ (STR) long_as_Date : (long) longdateformat {
    char dateStr[12];
    id newdate;

    newdate = [self new];
    [newdate day_number:longdateformat];
    strcpy(dateStr, [newdate convert_to_digit]);

    return dateStr;
}

// Answer a date that is an Integer number of days after the reciever
- (char *)addDays:(int) anInteger {
    long temp ;
    char * dayson ;
    temp = 86400 * anInteger + self->day_number ;
    dayson = ctime(&temp) ;

```



```

    return dayson ;
}

// subtractDays : Returns the date that is an integer number of
//                 days before the receiver date.
- (char *) subtractDays: (int) anInteger {
    long temp ;
    char * daysback ;

    temp = self->day_number - (86400 * anInteger) ;
    daysback = ctime(&temp) ;
    return daysback ;
}

// day : the number of days from the receiver to January 1 1970
- (int) day {
    int days_from_1970 ;
    days_from_1970 = self->day_number / 86400 ;
    return days_from_1970 ;
}

// dayIndex : Answer a number from 1 to 7 indicating the weekday
//            number of the receiver
- (int) dayIndex {
    int day_index ;
    day_index = (([self day] % 7) + 1) ;
    return day_index ;
}

// dayName : Answer the name of week day of the receiver
- (char *) dayName {
    //return day_name[[self dayIndex]] ;
    dateSt = localtime(&self->day_number) ;
    return day_name[dateSt->tm_wday + 1] ;
}

- (long) min: (long) magnitude {
    printf("min1:\n") ;
    if (self->day_number < magnitude)
        return self->day_number ;
    else
        return magnitude ;
}

// Year : Returns the year as an integer to receiver Date
- (int) year {
    dateSt = localtime(&self->day_number) ;
    return dateSt->tm_year + 1900 ;
}

// Month : Returns the month as an integer to receiver Date
- (int) month {
    dateSt = localtime(&self->day_number) ;
    return dateSt->tm_mon + 1 ;
}

// daysInMonth : Returns the number of days in the receiver month
- (int) daysInMonth {
    int leap = 0 ;
    int temp ;

    temp = [self month] ;
    if (temp == 2)
        leap = leapyears(dateSt->tm_year + 1900) ;
    return month_index[temp+1] - month_index[temp] + leap ;
}

// dayInMonth : Returns the dayInMonth as an integer to receiver Date

```

```

- (int) dayinMonth {
    dateSt = localtime(&self->day_number) ;
    return dateSt->tm_mday ;
}

// dayOfYear : Returns the number of days in the receivers year
- (int) dayOfYear {
    return month_index[[self month]] + [self dayinMonth] - 1 ;
}

// daysLeftInMonth : Returns the days remaining in the receivers month
- (int) daysLeftInMonth {
    int leap = 0;
    dateSt = localtime(&self->day_number) ;
    if (dateSt->tm_mon > 0)
    {
        leap = leapyears(dateSt->tm_year) ; }
    return month_index[(dateSt->tm_mon+1)] - month_index[dateSt->tm_mon]
    - [self dayinMonth] + leap ;
}

// monthIndex : Return a number from 1 to 12 indicating the month
//               of the receiver.
- (int) monthIndex {
    dateSt = localtime(&self->day_number) ;
    return dateSt->tm_mon + 1 ;
}

// monthName : Return a string representing the month name of
//             the receiver.
- (char *) monthName {
    return month_name[[self monthIndex]] ;
}

// subtractDate : Returns the number of days between the receiver
//               an aDate.
- (int) subtractDate: aDate {
    return [self day] - [aDate day] ;
}

// elapsedDaysSince : The number of elapsed days between the receiver and aDate
- (int) elapsedDaysSince: aDate {
    return [self day] - [aDate day] ;
}

// elapsedMonthsSince : The number of elapsed months between the receiver
//                     and aDate
- (int) elapsedMonthsSince: aDate {
    return((([self year] - [aDate year] * 12)
            + ([self month] - [aDate month])));
}

// elapsedSecondsSince : The number of elapsed seconds between the receiver
//                     and aDate.
- (int) elapsedSecondsSince: aDate {
    return self->day_number - aDate->day_number ;
} // check above

// firstDayInMonth : The number of the first day in the receiver month
//                   relative to the beginning of the receivers year.
- (int) firstDayInMonth {
    int leap ;

    dateSt = localtime(&self->day_number) ;
    if (dateSt->tm_mon > 1)
    {

```

```
    leap = leapyears(dateSt->tm_year) ;  
    return(month_index[dateSt->tm_mon] + leap) ;  
}  
else  
    return month_index[dateSt->tm_mon] ;  
}  
=;
```

```

// OBJECTIVE-C SOURCE FILE FOR THE CLASS "State";
// THIS CLASS CORRESPONDS TO A STATE IN AN AUTOMATA

#include "objc.h"
#include "wbs.h"
#include "cal.h"

@requires String;
@requires Sequence;
@requires OrdCltn;
@requires Event;
@requires Error;

=State : Object(CalGroup , Primitive, Collection)
{
// INSTANCE VARIABLES;
id event; // the user event;
id eventCollection; // all the state's events (instances of Event class);
id eventMessageCollection; // the messages to be displayed to the the user;
// To each event in eventCollection corresponds
// a message in eventMessageCollection

id error; // the error to display
id errorCollection; // the errors to be displayed (instances of Error class);
id errorMessageCollection; // the error messages to be displayed ;
// To each error in errorCollection corresponds
// a message in errorMessageCollection

id miscMessageCollection; // all other messages to display are grouped in this cltn;
id stateCollection; // all the states to which the state can transit
// to each event in eventCollection corresponds a state
// in stateCollection

id currentObject; // the object on which the user works
id automata ; // the automata in which the state is
id relatedClass; // the class on which the state operates
int typeOfResponse; // type of response is integer or string
int choiceInMenuResponse; // the user response in case of a menu
float floatResponse; // the user response in case of a question-answer
char stringResponse[MAX_SIZE_OF_NODE_NAME]; // the user response in case of a question-answer
}

// FACTORY METHODS;

// creates a new state and initialise it by default;
// returns the created state
+ new {
    id aState;
    aState = [super new];
    [aState initialise];
    return aState;
}

//INSTANCE METHODS;

// initialises the state by default.
// by default initialises the receiver errorCollection
// and errorMessageCollection with two errors
- initialise {
    [self errorCollection:
        [OrdCltn with:2, [Error new:NO_ERROR],
                        [Error new:OUT_OF_MENU_BOUNDS]]];
    [self errorMessageCollection:
        [OrdCltn with:2, NO_ERROR_MESSAGE,
                    OUT_OF_MENU_BOUNDS_MESSAGE]];
    return self;
}

- eventCollection: anEventCollection {

```

```
        eventCollection = anEventCollection;
        return self;
    }

    - eventCollection {
        return eventCollection;
    }

    - eventMessageCollection: aMessageCollection {
        eventMessageCollection = aMessageCollection;
        return self;
    }

    - eventMessageCollection {
        return eventMessageCollection;
    }

    - errorCollection: anErrorCollection {
        errorCollection = anErrorCollection;
        return self;
    }

    - errorCollection {
        return errorCollection;
    }

    - errorMessageCollection: anErrorMessageCollection {
        errorMessageCollection = anErrorMessageCollection;
        return self;
    }

    - errorMessageCollection {
        return errorMessageCollection;
    }

    - miscMessageCollection: aMiscMessageCollection {
        miscMessageCollection = aMiscMessageCollection;
        return self;
    }

    - miscMessageCollection {
        return miscMessageCollection;
    }

    - stateCollection: aStateCollection {
        stateCollection = aStateCollection;
        return self;
    }

    - stateCollection {
        return stateCollection;
    }

    -currentObject : aCurrentObject {
        currentObject = aCurrentObject;
        return self;
    }
```

```

- currentObject {
    return currentObject;
}

-automata : anAutomata {
    automata = anAutomata;
    return self;
}

- automata {
    return automata;
}

- relatedClass : aClass {
    relatedClass = aClass;
    return self;
}

- relatedClass {
    return relatedClass;
}

- typeOfResponse : (int) aTypeOfResponse {
    typeOfResponse = aTypeOfResponse;
    return self;
}

- (int) choiceInMenuResponse {
    return choiceInMenuResponse;
}

- (float) floatResponse {
    return floatResponse;
}

- (STR) stringResponse {
    return stringResponse;
}

// Displays the user messages.
// Test whether the type of response waited from the user
// is a choice in a menu or a question response
- display {
    id aSequence;
    id aMessage;
    int iMax;
    int i;

    iMax = [eventMessageCollection size];
    // 1 - if it is a choice in a menu display the
    // menu items found in eventMessageCollection
    if ((iMax >= 1) && (typeOfResponse == CHOICE_IN_MENU)) {
        for (i = 0; i < iMax ; i++)
            printf("%d - %s \n",i+1,[[eventMessageCollection at:i] str]);
        printf("Your choice ?");
    }
}

```

```

    }
    // 2 - if it a question response displays the
    //      only item found in eventMessageCollection
    else if ((iMax == 1) &&
              ((typeofResponse == STRING_RESPONSE) ||
               (typeofResponse == FLOAT_RESPONSE))) {
        printf("%s ?\n", [[eventMessageCollection at:0] str]);
        printf("?");
    }
    else
    // 3 - if there is no eventMessageCollection
    //      generates an error
        [self error:[NOT_INITIALISED_STATE_MESSAGE str]];
}

// returns the error which number is anErrorNumber
- findError: (int) anErrorNumber {
    id anError;
    id aSequence;
    aSequence = [errorCollection eachElement];
    while (anError = [aSequence next])
        if ([anError number] == anErrorNumber)
            return anError;
}

// returns the event which number is anEventNumber
- findEvent: (int) anEventNumber {
    id anEvent;
    id aSequence;
    aSequence = [eventCollection eachElement];
    while (anEvent = [aSequence next])
        if ([anEvent number] == anEventNumber)
            return anEvent;
}

// returns the error offset in errorCollection which
// number is anErrorNumber
- (int) findErrorOffset: (int) anErrorNumber {
    id theError;
    theError = [self findError:anErrorNumber];
    return [errorCollection offsetOf: theError];
}

// returns the event offset in eventCollection which
// number is anEventNumber
- (int) findEventOffset: (int) anEventNumber {
    id theEvent;
    theEvent = [self findEvent:anEventNumber];
    return [eventCollection offsetOf: theEvent];
}

// reads the user response and assign the variables
// choiceInMenuResponse or floatResponse or stringResponse
- read {
    switch (typeofResponse) {
        case CHOICE_IN_MENU :
            scanf("%d", &choiceInMenuResponse);
            break;
        case FLOAT_RESPONSE :
            scanf("%f", &floatResponse);
            break;
    }
}

```

```

        case STRING_RESPONSE :
            scanf("%s", stringResponse);
            break;
        }
        return self;
    }

// test whether the user response in case of a menu
// is inside the menu bounds
- correct {
    return self;
}

// displays the error message which corresponds to the
// instance variable error
- errorMessage {
    int offset;
    offset = [errorCollection offsetOf: error];
    printf("%s \n", [[errorMessageCollection at: offset] str]);
    return self;
}

- treat {
    if (typeOfResponse == CHOICE_IN_MENU)
        event = [eventCollection at: (choiceInMenuResponse -1)];
    else if ((typeOfResponse == STRING_RESPONSE) ||
             (typeOfResponse == FLOAT_RESPONSE))
        event = [eventCollection firstElement];
    else;
    return self;
}

- followingState {
    int offset;
    id aFollowingState;
    if ((stateCollection != nil) &&
        ([event number] != NO_EVENT)) {
        offset = [eventCollection offsetOf: event];
        aFollowingState = [stateCollection at: offset];
        [[aFollowingState automata] currentState: aFollowingState];
        [aFollowingState execute: currentObject];
    }
    return self;
}

- treatG : (int) eventnumber {
    event = [eventCollection at: eventnumber];
    return self ;
}

- execute : aCurrentObject {
    if ([aCurrentObject notEqual: nil])
        [self currentObject: aCurrentObject];
    do {
        error = [self findError: NO_ERROR];
        [self display];
        [self read];
        [self correct];
        if ([error number] != NO_ERROR)
            [self errorMessage];
    }
}

```



```
    }  
    while ([error number] != NO_ERROR);  
    [self treat];  
    [self followingState];  
}  
=;
```

```

// Objective-C source file for the class CalAutomata
#include "objc.h"
#include "wbs.h"
#include "cal.h"

@requires State;
@requires CalInitialState;
@requires CalSecondState;
@requires CalThirdState ;
@requires OrdCltn;

=CalAutomata : Object(CalGroup , Primitive, Collection)
{
//INSTANCE VARIABLES;
id firstState;          // the initial state of the automata;
id secondState;          // the node operations state of the automata;
id thirdState ;
id currentState;        // the state of the automata being executed;
id automata;             // the automata in which the automata is contained
}

//INSTANCE METHODS;

// To initialise the automata:
// - initialises the states
// - define each state
- initialise {
    [self initialiseStates];
    [self defineFirstState];
    [self defineSecondState];
    [self defineThirdState] ;
}

// initialises each state
- initialiseStates {
    firstState = [CalInitialState new];
    [firstState automata:self];
    secondState = [CalSecondState new];
    [secondState automata:self];
    thirdState = [CalThirdState new] ;
    [thirdState automata:self] ;
    return self
}

// defines the states reachable from the first state
- defineFirstState {
    id anOrdCltn;
    anOrdCltn = [OrdCltn with: 4, firstState,firstState,
                      firstState,UNDEFINED_STATE];
    [firstState stateCollection: anOrdCltn];
}

// defines the states reachable from the second state
- defineSecondState {
    id anOrdCltn;
    anOrdCltn = [OrdCltn with: 4, secondState,          //1
                      secondState,                        //2
                      thirdState,                          //3
                      UNDEFINED_STATE];
}

```

```

        [secondState stateCollection: anOrdCltn];
    }

- defineThirdState {
    id anOrdCltn ;
    anOrdCltn = [OrdCltn with: 6, thirdState, //1
                thirdState, //2
                thirdState, //3
                thirdState, //4
                thirdState, //5
                secondState] ;
    [thirdState stateCollection: anOrdCltn];
}

// lunches the automata current state if exist
// else lunches the automata first state
- execute : anObject {
    if (currentState != nil)
        [currentState execute:[currentState currentObject]];
    else
        [firstState execute: anObject];
    return self
}

// puts aState as the current state of the receiver
// and all the receiver super automata
- currentState: aState {
    currentState = aState;
    if (automata != nil)
        [automata currentState: aState];
    return self;
}

- currentState {
    return currentState;
}

- currentObject {
    return [currentState currentObject];
}

- automata: anAutomata {
    automata = anAutomata;
    return self;
}

- automata {
    return automata;
}

- firstState {
    return firstState;
}

```

```
- secondState {  
    return secondState;  
}  
  
- thirdState {  
    return thirdState ;  
}  
=:
```

```

// Objective-C source file for the class CalInitialState
#include "objc.h"
#include "cal.h"
#include "wbs.h"
#include "date.h"
#include "sac_global.h"

@requires IS;
@requires SortCltn;
@requires TaskEvent;
@requires String;
@requires OrdCltn;
@requires Event;
@requires Date ;
@requires Graphic;

extern id anEventDataCltn;
extern id aCALload;

#define daysec (long) 86400

extern      Gevicocar_t      confirmico,abortico;
extern      Gevicocar_t      voidico;
extern      Gevicocar_t      notitle;
extern      Gevsgrgph_t      screen;
extern      Gevlift_t        voidlift;
extern      Gevtab_t          confirmtab,conftoptab;
extern      Gevtab_t          caltoptab;
extern      Gevwdw_t          currentwdw;
extern      Gevrow_t          someobject;
extern      Matrix_t          voidmatrix;
extern      Matrix_t          confirm_mat;
extern      Matrix_t          cal_mat;
extern      int               endofjob;

Matrix_t    caltab;

Gevicocar_t  calleft,calright,calendarico;
Gevicocar_t  opcico;
Gevicocar_t  intico;
Gevicocar_t  tasico;
Gevrow_t     caltop,calwdw;
Gevtab_t     caltabmid;
Gevvt_t      evt;

int eventnum ;
long datescale ;
id dateCltn, dateStrCltn , taskCltn;
= CalInitialState : State (CalGroup , Primitive,Collection)
{
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;
    id aMiscMessCltn;
    dateCltn = [OrdCltn new:35] ;
    dateStrCltn = [OrdCltn new:35];

    // inherits its superclass initialisation
    [super initialise];

    taskCltn = [OrdCltn with: 5,    "Imports :",
                "Products :",
                "Meetings :",
                "Personnel :",

```

```

        "Work in Progress :");

    // initialise the menu messages
    aMessCltn = [OrdCltn with: 4, OPEN_CALENDAR_MESSAGE,
                SET_INTERVAL_MESSAGE,
                SET_TASK_MESSAGE,
                QUIT_MESSAGE];
    [self eventMessageCollection: aMessCltn];

    // initialise the corresponding events
    anEventCltn = [OrdCltn with:4, [Event new:OPEN_CALENDAR_EVENT],
                    [Event new:SET_INTERVAL_EVENT],
                    [Event new:SET_TASK_EVENT],
                    [Event new:QUIT_EVENT]];
    [self eventCollection: anEventCltn];

    // this is a menu
    [self typeOfResponse:CHOICE_IN_MENU];
}

- open_cal {
    long startCal ;
    BOOL VALID = NO;

    datescale = 86400 ;
    VALID = [Graphic estDate];
    if (VALID) {
        [Date getdates];
        [Graphic calDisplay];
    }
    return self ;
}

- set_interval {
    int days ;
    BOOL VALID = NO;

    days = [Graphic datescale];
    if (days != 0) {
        datescale = days * 86400;
        VALID = [Graphic estDate];
        if (VALID) {
            [Date getdates];
            [Graphic calDisplay];
        }
    }
    return self ;
}

- set_task {
    id aCltn;
    id anEventDataStrCltn;
    id aTaskEvent;
    id anOrdCltn;
    id taskSeq;
    id bTask;
    int i;
    BOOL VALID = NO;

    aCltn = [aCALload taskCltn];
    anEventDataStrCltn = [OrdCltn new];
    for(i=0; i < [aCltn size]; i++)
        [anEventDataStrCltn addIfAbsentMatching:[aCltn at:i] startDate]];
    [anEventDataStrCltn print];
    anEventDateCltn = [SortCltn orderedBy:"taskcompare:" onDups:1] ;
    for (i=0; i < [anEventDataStrCltn size]; i++) {
        aTaskEvent = [TaskEvent create:[anEventDateStrCltn at:i]];
    }
}

```

```

        [anEventDataCltn add:aTaskEvent];
    }
    anOrdCltn = [anEventDataCltn asOrdCltn];
    dateStrCltn = [OrdCltn new];
    taskSeq = [anOrdCltn eachElement];
    while (bTask = [taskSeq next])
        [dateStrCltn add:[bTask eventData] str]];

    [Graphic calTaskDisplay];
}

-display {
    [Graphic datescrsel];
}

- read { // stay with window until confirmed yes
    // if no selection is made open_cal() default is used
    eventnum = 0;
    endofjob = GEV_FALSE;
    while ( endofjob != GEV_TRUE) {
        evt = Gevscrph_wait_event(screen);

        if (Gevvt_get_wdw(evt) == Gevwdw_get_wdw(currentwdw)) {
            if (Gevvt_get_ttyp(evt) == GEV_SELECTED) {
                if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(cal_mat,1,1))) {
                    eventnum = 0;
                    Gevchr_deselect(Matrix_entry(cal_mat,2,1));
                    Gevchr_deselect(Matrix_entry(cal_mat,3,1));
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(cal_mat,2,1))) {
                    eventnum = 1;
                    Gevchr_deselect(Matrix_entry(cal_mat,1,1));
                    Gevchr_deselect(Matrix_entry(cal_mat,3,1));
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(cal_mat,3,1))) {
                    eventnum = 2;
                    Gevchr_deselect(Matrix_entry(cal_mat,1,1));
                    Gevchr_deselect(Matrix_entry(cal_mat,2,1));
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(confirm_mat,1,1))) {
                    if (eventnum == 0) {
                        Gevscrph_rem_window(screen,currentwdw);
                        endofjob = GEV_TRUE;
                    }
                    else if (eventnum == 1) {
                        Gevscrph_rem_window(screen,currentwdw);
                        endofjob = GEV_TRUE;
                    }
                    else if (eventnum == 2) {
                        Gevscrph_rem_window(screen,currentwdw);
                        endofjob = GEV_TRUE;
                    }
                }
                else if (Gevobj_eq(Gevvt_get_botobj(evt),Matrix_entry(confirm_mat,1,2))) {
                    eventnum = 3;
                    endofjob = GEV_TRUE;
                }
            }
        }
    }

    - treat {
        dateCltn = [OrdCltn new] ;
        [super treatG:eventnum];
    }
}

```

```
        switch ([event number]) {
case OPEN_CALENDAR_EVENT :
    [self open_cal] ;
    break ;
case SET_INTERVAL_EVENT :
    [self set_interval];
    break;
case SET_TASK_EVENT :
    [self set_task];
    break;
case QUIT_EVENT :
    [IS interpret:"QUIT"];
    exit(0);
    break;
}
return self;
}
=;
```



```

// Objective-C source file for the class CalSeceondState
#include "objc.h"
#include "wbs.h"
#include "cal.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence;
@requires Date ;

=CalSecondState : State(CalGroup , Primitive, Collection)
{
//INSTANCE VARIABLES;
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;

    // inherits its superclass initialisation
    [super initialise];

    // initialise the events

    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:FORWARD_EVENT]];
    [anEventCltn add: [Event new:BACKWARD_EVENT]];
    [anEventCltn add: [Event new:VIEW_EVENT]];
    [anEventCltn add: [Event new:QUIT_EVENT]];
    [self eventCollection: anEventCltn];

    // initialise the messages

    aMessCltn = [OrdCltn new];
    [aMessCltn add: FORWARD_MESSAGE];
    [aMessCltn add: BACKWARD_MESSAGE];
    [aMessCltn add: VIEW_MESSAGE];
    [aMessCltn add: QUIT_MESSAGE];
    [self eventMessageCollection: aMessCltn];

    // it is a menu
    [self typeOfResponse:CHOICE_IN_MENU];
}

- goForward {
    [Date forward] ;
    [Date dateDisplay] ;
    return self;
}

- goBackward {
    [Date back] ;
    [Date dateDisplay] ;
    return self;
}

- view {
    return self;
}

// Calls the treatment corresponding to the event
- treat {

```

```
[super treat];  
  
switch ([event number]) {  
  
case FORWARD_EVENT :  
    [self goForward];  
    break;  
  
case BACKWARD_EVENT :  
    [self goBackward];  
    break;  
  
case VIEW_EVENT :  
    [self view];  
    break;  
  
case QUIT_EVENT:  
    exit(0);  
    break;  
}  
return self;  
}  
=;
```

```

// Objective-C source file for the class CalThirdState
#include "objc.h"
#include "wbs.h"
#include "cal.h"

@requires Error;
@requires Event;
@requires String;
@requires OrdCltn;
@requires Sequence;

= CalThirdState : State(CalGroup , Primitive, Collection)

{
//INSTANCE VARIABLES;
}

//INSTANCE METHODS;

- initialise {
    id aMessCltn;
    id anEventCltn;

    // inherits its superclass initialisation
    [super initialise];

    // initialise the events

    anEventCltn = [OrdCltn new];
    [anEventCltn add: [Event new:IMPORTS_EVENT]];
    [anEventCltn add: [Event new:PRODUCTS_EVENT]];
    [anEventCltn add: [Event new:MEETING_EVENT]];
    [anEventCltn add: [Event new:PERSONNEL_EVENT]];
    [anEventCltn add: [Event new:WORK_IN_PROGRESS_EVENT]] ;
    [anEventCltn add: [Event new:EXIT_EVENT]];
    [self eventCollection: anEventCltn];

    // initialise the messages

    aMessCltn = [OrdCltn new];
    [aMessCltn add: IMPORTS_MESSAGE];
    [aMessCltn add: PRODUCTS_MESSAGE];
    [aMessCltn add: MEETING_MESSAGE];
    [aMessCltn add: PERSONNEL_MESSAGE];
    [aMessCltn add: WORK_IN_PROGRESS_MESSAGE];
    [aMessCltn add: EXIT_MESSAGE];
    [self eventMessageCollection: aMessCltn];

    // it is a menu
    [self typeOfResponse:CHOICE_IN_MENU];
}

- import_events {
    return self ;
}

- product_events {
    return self ;
}

- meeting_events {
    return self ;
}

- personnel_events {

```

```
    return self ;
}

- work_in_progress_events {
    return self ;
}

// Calls the treatment corresponding to the event

- treat {

    [super treat];

    switch ([event number]) {

    case IMPORTS_EVENT :
        [self import_events];
        break;

    case PRODUCTS_EVENT :
        [self product_events];
        break;

    case MEETING_EVENT :
        [self meeting_events];
        break;

    case PERSONNEL_EVENT :
        [self personnel_events];
        break;

    case WORK_IN_PROGRESS_EVENT :
        [self work_in_progress_events];
        break;

    case QUIT_EVENT:
        break;
    }
    return self;
}

=;
```

```
// Objective-C source file for the class Task

= Task : Object ( CalGroup,Collection, Primitive )
{
    id name;
    id startDate;
    id endDate;
    id product;
    id consumedCltn;
    id workCltn;
}

- taskName {
return name;
}

- taskName: aString {
name = aString;
}

- startDate {
return startDate;
}

- startDate: aDate {
startDate = aDate;
}

- endDate {
return endDate;
}

- endDate: aDate {
endDate = aDate;
}

- product {
return product;
}

- product: aString {
product = aString;
}

- consumedCltn {
return consumedCltn;
}

- consumedCltn: aCltn {
consumedCltn = aCltn;
}

- workCltn {
return workCltn;
}

- workCltn: aCltn {
workCltn = aCltn;
}
=;
```

```
// Objective-C source file for the class Work

= Work : Object ( CalGroup, Collection, Primitive )
{
    id fromDate;
    id toDate;
    id resourceName;
}

- resourceName {
return resourceName;
}

- resourceName: aString {
resourceName = aString;
}

- fromDate {
return fromDate;
}

- fromDate: aDate {
fromDate = aDate;
}

- toDate {
return toDate;
}

- toDate: aDate {
toDate = aDate;
}
=;
```

Bibliography

- AFSC87** AFSC Pamphlet 800-XX, *Software Risk Management*, DEPT. OF THE AIR FORCE(FRANCE).
- SPE 88** Computer Associated International,Inc., *Super Project Expert*, .
- Alt 78** Alter,S and Ginzberg,M., *Managing Uncertainty in MIS Implementation*, SLOAN MANAGEMENT REVIEW,FALL,1978,PP.23-31.
- Ara 88** Arapis,G and Kappel,G., *Organizing Objects in an Object Software Base*, UNIVERSITY OF GENEVA.
- Bar 86** Barth,P S., *An object-oriented approach to graphic interfaces*, ACM TRANSACTIONS ON GRAPHICS, VOL.5,NO.2,PP142-172,APRIL 86.
- Ben 86** Benyon,D. and Skidmore,S., *Towards a Tool Kit for the Systems Analyst*, THE COMPUTER JOURNAL VOL.30, No.1 PP2-7.
- Boe 81** Boehm,B., *Software Engineering Economics*, PRENTICE-HALL.
- Boo 86** Booch,Grady., *Object-Oriented Development*, SOFTWARE ENGINEERING, VOL. SE-12, No.2 FEBURARY 1986.
- Bud 87** Bud, Timothy A., *A Little Smalltalk*, ADDISION-WESLEY 1987.
- Car 84** Cardelli,Luca., *Semantics of Multiple Inheritance*, SEMANTICS OF DATA TYPES. SPRINGER-VERLAG 1984.
- Car 85** Cardelli,Luca. and Wegner,Peter., *On Understanding Types, Data Abstraction, and Polymorphism*, COMPUTING SURVEYS, VOL. 17, No. 4, DEC. 1985.
- Cha 85** Chapman,C., *Select an Approach to Project Time and Cost Planning*, PROJECT MANAGEMENT,VOL.3(1),FEB 1985.
- Cou 86** Coutaz,Joelle, *The Construction of User Interfaces & the Object Paradigm*, ECOOP87.
- Cox 85** Cox, Brad, *Object-Oriented Programming: An Evolutionary Approach*, ADDISION-WESLEY 1986.
- Cox 85** Cox,Brad. & Ledbetter,Lamar., *Software-IS'c*, BYTE MAZAGINE JUNE 1985.
- Coxb86** Cox,B. and Hunt,B., *Objects,Icons, and Software-ICs*, BYTE MAZAGINE PP.161-176 AUGUST 1986.
- Dah 66** Dahi,Ole-Johan. & Nygaarg,Kristen., *SIMULA - An Algol-based Simulation*, COMMUNICATIONS OF THE ACM, VOL.9, No.9,PP.671-678, SEPT 1966.

- Dan 89 Daniels, John., *The Emergence of Object-Oriented Methods*, REX SYSTEMS.
- Dav 82 Davis, G.B., *Strategies for Determining Requirements*, IBM SYSTEM JOURNAL, VOL.21(1), 1982, PP.4-30.
- DeM 80 DeMarco, T., *Structured Analysis: System Specification*, YOURDAN 1980.
- Dod 89 Dodani, Mahesh H. Hughes, Charles E. & Moshell, Micheal J., *Seperation of Powers*, BYTE MAZAGINE MARCH 1989.
- Dra 89 Drake, Richard., *Object-Oriented Programming in C++*, PERSONAL COMPUTER WORLD MAZAGINE FEBRUARY 1989.
- Ger 82 Gersting, Judith L., *Mathematical Structures for Computer Science*, FREEMAN 1982.
- Gol 83 Goldberg, Adele. and Robson, David, *Smalltalk-80: The language and Its Implementation*, ADDISION-WESLEY 1983.
- Gol 84 Goldberg, Adele, *Smalltalk-80: The Interactive Programming Environment*, ADDISION-WESLEY 1984.
- Hal 88 Halbert, Daniel C. and O'Brien, Patrick, *Using Types and Inheritance in Object Orient Programming*, DIGITAL EQUIPMENT CORP..
- Hen 68 Hennie, Fredrick C., *Finite-State Models for Logical Machines*, WILEY 1968.
- Hop 79 Hopcroft, John E. & Ullman, Jeffrey D., *Introduction to Automata Theory, Languages, and Computation*, ADDISON-WESLEY 1979.
- Hor 87 Horn, Chris., *Conformance, Genericity, Inheritance and Enhancement*, ECOOP87.
- Jac 83 Jackson, M., *System Development*, PRENTICE-HALL.
- Jac 87 Jacobson, Ivar, *Object Oriented Development in an Industrial Environment*, OOPSLA87.
- Joh 88 Johnson, Ralph E. & Foote, Brian., *Designing Reusable Classes*, JOURNAL OF OBJECT ORIENTED PROGRAMMING JUNE/JULY 1988.
- Kae 86 Kaehler, Ted. and Patterson, Dave., *A Taste of Smalltalk*, NORTON 1986.
- Kee 81 Keen, J., *Managing System Development*, WILEY 1981.

- Kra 83 Krasner, Glenn., *Smalltalk-80: Bits of History, Words of Advice*, ADDISON-WESLEY 1983.
- Lie 85 Lieberman, H., *There's More to Menu Systems than Meets the Screen*, SIGGRAPH85, 19(3), 181-189.
- Lis 87 Liskov, Barbara., *Date Abstracyion and Hierarchy*, OOPSLA87.
- Mac 82 MacLennan, B., *Values and objects in programming languages*, SIGPLAN NOTICES. VOL. 17, No. 12, p. 75, DEC. 1982.
- Mel 87 Mellor, Stephen J., *Object-Oriented Programming and Other Advanced Techniques*, CRAI SPRING INTERNATIONAL SEMINAR.
- Mey 87 Meyer, B., *Reusability: The Case for Object Oriented Design*, IEEE SOFTWARE MARCH 1987, PP. 50-64.
- Mey 87 Meyer, B., *Reusability: The Case for Object-Oriented Design*, IEEE SOTFWARE, MARCH 1987, PP50-64.
- Mey 88 Meyer, Bertrand., *Object-Oriented Software Construction*, PRENTICE HALL 1988.
- Min 67 Minsky, Marvin., *Computation: Finite and Infinite Machines*, PRENTICE HALL 1967.
- Moy 89 Moynihan, T. McCloskey, G. Verbruggen, R., *RISKMAN1: A Prototype Tool for Risk Analysis*, CASE89 WORKING PAPER CA-0389.
- Nie 87 Nierstrasz, O.M., *A Survey of Object-Oriented Concepts*, ACTIVE OBJECT ENVIRONMENT UNIVERSITY OF GENEVA PP1-17.
- OC 87 The StepStone Corporation, *Objective-C Ver. 3.3 Reference Manual*,
- Pas 86 Pascoe, Geffory A., *Elements of Object Oriented Programming*, BYTE MAZAGINE AUGUST 1986.
- Ren 82 Rentsch, T., *Object-Oriented Programming*, SIGPLAN NOTICES VOL. 17, NO. 9, P. 51, SEPT. 1982.
- Rob 81 Robson, David., *Object-Oriented Software Systems*, BYTE MAGAZINE VOL. 6, No. 8, AUG. 1981.
- Rob 89 Robinson, Peter., *Hierarchic Object Oriented Design - HOOD*, EUROPEAN SPACE AGENCY.
- Sch 86 Schmucker, Kurt J., *Object-Oriented Languages for the Machintosh*, BYTE MAGAZINE AUGUST 1986.

- Schb86 Schmucker,K J., *MACAPP: An Application Framework*, BYTE MAZAGINE PP189-193 AUGUST 1986.
- Ste 86 Stefilk,Mark. & Bobrow, Daniel g., *Object-Oriented Programming: Themes and Variations*, AI MAGAZINE VOL. 6, NO. 4, WINTER 1986.
- Str 86 Stroustrup,Bjarne., *The C++Programming Language*, ADDISON-WESLEY 1986.
- Sun 85 Sun Microsystems, *Command Reference Manual*, SUN MICROSYSTEMS INC.
- Syn 86 Synder,A., *Encapsulation and Inheritance in Object-Oriented Programming*, OOPSLA86 VOL.21, No.11,PP.38-45,Nov.1986.
- Tho 89 Thomas,Dave., *What's in an Object*, BYTE MAZAGINE MARCH 1989.
- Tsi 87 Tsichritzis,D. & Nierstrasz,O., *Application Development Using Objects*, ACTIVE OBJECT ENVIRONMENTS, UNIVERSITY OF GENEVA PP.18-30.
- Ver 88 Verbruggen,Renaat., *Object Oriented Design - Does it Exist?*, CASE88.
- Was 88 Wasserman,Anthony I. Pircher,Peter A. & Muller,Robert J., *An Introduction to Object-Oriented Structured Design*, INTERACTIVE DEVELOPMENT ENVIRONMENTS, INC..
- Weg 87 Wegner,Peter., *The Object Oriented Classification Paradigm*, PRENTICE HALL 1987.
- Weg 88 Wegner,Peter., *Dimension of Object Based language Design*, BROWN UNIVERSITY.
- Wie 88 Wiener,Richard S. and Lewis,j Pinson, *An Introduction to Object-Oriented Programming and C++*, ADDISON-WESLEY.
- Woo 82 Wood-Harper,A T., *A Tazonomy of Current Approaches to Systems Analysis*, THE COMPUTER JOURNAL 1982.
- Zal 77 Zaltman,f. and Duncan,G., *Strategies for Planned Change*, WILEY,1977.