x-21-159091-5
F6982

ENHANCING DISCRETE EVENT MODELLING BY INTERFACING EXPERT SYSTEMS AND SIMULATION MODELS

By

Daniel Goodman

B.Sc. (LSE)

Thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at the London School of Economics and Political Science, University of London.

May 1992.

ABSTRACT

This thesis investigates the representation of operational decision makers within simulation modelling.

Artificial Intelligence concepts, such as expert systems focus on the problem of representing, in high-level code, complex real-world decision making problems.

The author therefore proposes that the use of expert system technology may provide an improved means of representing operational decision tasks and that as a consequence, apriori possibilities may exist in the context of model experimentation based on alternative operational policies.

The thesis further investigates the nature of operational decision making and the potential need to represent within a model, inter-dependencies between decision makers.

A prototype system called ESSIM is developed which comprises of two interlinked components, a discrete event simulation module and expert system module. The benefits of the proposed approach are then assessed by comparing the functionally of ESSIM with conventional modelling techniques. The comparison is carried out by developing three alternative models of an automated container port, one of these using ESSIM. Experiments were then devised and executed which seek to draw conclusions on the thesis proposal.

ACKNOWLEDGEMENTS

Many thanks to David Balmer, my supervisor, for his invaluable guidance in producing this thesis.

I would also like to thank my wife, parents, in laws, and friends for their moral support.

<u>CONTENTS</u>

CHAPTER FOUR: ESSIM - AN ENVIRONMENT FOR SIMULATION

CHAPTER FIVE: VALIDATION OF ESSIM USING A CONTAINER PORT MODEL

<u>CHAPTER SIX</u>: CONCLUSION

APPENDICES:

CHAPTER ONE

INTRODUCTION

1.1 THE THESIS PROPOSITION

This thesis will investigate possible approaches in using Artificial Intelligence techniques in improving the representation of operational decision makers within simulation models.

The thesis proposition is that expert systems techniques may provide an improved means of representing within the model, operational policies which in the real-world dictate the course of events. Such operational policies may require the involvement of multiple decision makers and may potentially involve the representation of some form of hierarchical management structure.

The belief that expert system technology may have a role to play within conventional simulation modelling is a consequence of the fact that much of Artificial Intelligence research is focused on providing tools for the resolution of complex real-world decision making tasks.

There are a number of potential benefits which could be derived from using Artificial Intelligence techniques in the detailed representation of operational decision makers and their inter-relationships. The main advantage is likely to result from the ease with which model experiments could be carried out based on alternative operational policies. (In the context of this thesis, the term "Model Adaptability" will be used to describe this benefit). A second derivative benefit is that adding model detail in the context of operational decision making, may ultimately result in a model which is a better representation of the real-world problem. The term, "Model Accuracy" will be used in this thesis to describe this benefit.

Observations similar to the above have already been made by a number of authors including Fishman[1973]. They assert that conventional simulation languages are not well suited to the representation of decision tasks. Several authors have identified the potential of Artificial Intelligence (AI) approaches to overcoming these difficulties. The possibility of integrating a model of operational decision-making in the form of an expert system and a conventional simulation model has been envisaged by O'Keefe and Roach[1987]. Flitman and Hurrion [1987] then provide the first practical insight into the potential of linking an Artificial Intelligence tool with a simulation model by building a system based on two inter-communicating micro-computers. The research presented in this thesis follows on and builds upon Flitman's [1986] pioneering work by concentrating on two key problems: (1) The representation of operational policies which are reflected by the real-world operational staff and their cooperative actions. (2) The need to create a "practical" modelling environment in which the link between simulation model and expert system is almost seamless.

1.2 THE RESEARCH STRATEGY

The first stage of the research involves a detailed literature study covering both simulation modelling and artificial intelligence. The emphasis of the literature study is in identifying appropriate state-of-the-art technology which could be applied in creating a simulation environment incorporating artificial intelligence techniques. Of particular interest are the various Artificial Intelligence approaches to the representation of "Knowledge" and the inference of conclusions from this knowledge. These can broadly be divided into AI languages (e.g. Lisp & Prolog) and AI Tools (e.g. Expert Systems and Object Oriented environments). Another important aspect of the literature study, is to learn what other researchers have achieved or proposed in the context of combining artificial intelligence and simulation modelling techniques. Finally, much of this thesis is concerned with the representation of decision making activities and the inter-relationship between decision makers during the process of enacting operational policies. Consequently, background research was necessary into the nature of decision making and the implications of hierarchical management structures.

The approach adopted in this thesis, was to build upon earlier work undertaken by Flitman[1986] and to investigate through the development of a number of prototype systems, the implications of integrating an expert system model of operational decision making and a conventional simulation model.

The development of the prototype environment involves the identification and resolution of the many practical difficulties in bringing together expert system and conventional simulation modelling techniques. The first implementation of a prototype environment principally served the purpose of identifying these practical difficulties and defining the specification of a second implementation. This second implementation seeks to overcome the problems identified in the first version and forms the basis of a generic simulation modelling environment within which the practical modelling experience could be obtained.

A research strategy based on the development of prototypes is only effective if one is able to define a means of comparing the value of the new simulation technique with a more conventional modelling approach. The research therefore includes the development of three simulation models of a container port. One using the proposed approach, another using conventional simulation techniques, and a third using Pascal functions to replicate some of the characteristics of an expert system. A number of experiments are then devised which seek to assess the functionality of the three models against the identified potential benefits of the proposed modelling environment.

1.3 RESEARCH BACKGROUND

The motivation for the proposed research originally arose from the author's involvement in the Computer Aided Simulation Modelling (C.A.S.M) group at the London School of Economics which brought together a number of research studies in simulation techniques.

One such project which was subsequently to provide a practical modelling context for this thesis was a joint project between the London School of Economics (L.S.E.) and the Instituto Nacional de Tecnologia (I.N.T.), a Brazilian research Centre. The intention was to produce computer simulation models for production planning and control which were to be used in assessing the consequences of different jobbing and batch production structures in a typical job-shop environment. I.N.T. were to provide the necessary expertise in production engineering whilst in the final stage of the research, two manufacturing concerns, NATEC LTDA and DANCOR S.A., were to provide the practical context. The computer systems used at the proposed sites were to be integrated with the models in providing a decision support system which was to aid management to schedule and control production.

The rationale in providing such a decision support tool was based on the fact that in a typical batch manufacturing environment, more than 90% of time is spent idle in queues awaiting processing. Consequently, it was felt that there was considerable room for improvements in productivity by rationalising the material flows.

Regrettably, geographical barriers led to difficulties in maintaining close contact with our Brazilian counterparts. The first prototype system encompassing an expert system and conventional simulation model was developed with the job-shop modelling requirements specifically in mind. However, the subsequent difficulty in gaining access to real-world experts meant that a different model had to be developed for the evaluation of the second prototype system. The new model was that of an automated and partially un-manned container port which was to be built by Highland Participants PLC on the Isle of Grain. The real-world container port was to be controlled by a software application, the core of which would consist of decision rules formalised within a 'knowledge-base'.

1.4 STRUCTURE OF THE THESIS

Chapter two consists of a review of the two areas of research appropriate to this thesis, simulation modelling and Artificial Intelligence. The chapter begins with an analysis of simulation modelling covering its purpose, limitations and applicability within decision support systems. A similar approach is taken in investigating the area of Artificial Intelligence though particular emphasis is placed on knowledge representation, a topic particularly pertinent to this thesis. A study of recent papers follows, outlining what are currently considered to be "advanced" systems in the area of co-operative systems involving both simulation and AI.

Chapter three focuses on the characteristics of decision making and seeks to identify the possible ways of representing these within simulation modelling and Artificial Intelligence. The possible ways of combining simulation and Artificial Intelligence knowledge representations are identified and the benefits and limitations critically compared. A choice is ultimately made as to the approach to be selected for the purpose of building a prototype system.

Chapter four describes the process applied in developing the prototype system referred to as ESSIM (Expert System SIMulation). The first prototype system which consisted of a model of a Job-shop is described and the conclusions drawn from this initial investigation outlined. The general design of the second prototype system is then explained followed by an in-depth definition of the function of the various modules of the system.

Chapter five outlines the development of the un-manned container port model already referred to in section 1.3. The implementation of this model forms the basis of the validation of the proposed methodology based on the link between simulation system and expert system. The structure of the simulation system component of the port model is explained followed by the expert system knowledge-base component. The process of experimenting with the port model is assessed with respect to the introduction of modifications to knowledge base and simulation system code. Particular emphasis is placed during this assessment process on the impact of the expert system approach on modelling aspects including model "Accuracy" and "Adaptability". The value of the new modelling process as encompassed in ESSIM is then compared with more conventional approaches through the implementation of the same port model code using existing modelling tools. The benefits and limitations of the ESSIM approach to modelling are then summarised.

Chapter six concludes the thesis by summarising the work undertaken and conclusions presented. The achievements of the research are formalised and suggestions made as to future work which could be undertaken in order to build upon the experiences that resulted from the research encompassed in this thesis.

CHAPTER TWO

RESEARCH CONTEXT

2.1 INTRODUCTION

The research presented in this thesis covers two distinct areas of knowledge, simulation and artificial intelligence. The background literature study presented and discussed in this chapter therefore commences with a review of the nature, goals and limitations of each of these technologies.

The literature study revealed that there existed some degree of overlap between simulation modelling and expert systems approaches. A number of published papers were also found which argued this case. Chapter two therefore continues to investigate the similarities between simulation modelling and expert systems and explores the work of other researchers who have attempted to carry out simulation modelling using artificial intelligence languages and tools.

Researchers have for some time been investigating the potential of using simulation and AI to mutual benefit. From the simulation perspective, AI provides the necessary tools for creating advisory systems to assist the user in all stages of the process of developing and experimenting with simulation models. From the AI perspective, simulation provides the required framework for handling problems involving temporal reasoning (time handling) and gives advisory systems the capability of investigating the future. Such research, may provide valuable experience and insight into knowledge representation techniques and methods used in bringing together simulation and AI. The chapter therefore concludes with a review of research studies which have investigated the interfacing of simulation models and AI languages and tools.

2.2 SIMULATION MODELLING

Computer simulation modelling dates back to the early days of computers. Nance[1981] and Shannon[1986] broadly divide the development of simulation into five stages. Up until the 1960's, simulation models were mostly coded directly in FORTRAN. In the early 60's, the concept of simulation modelling attracted much interest which spurred on the development of simulation specific languages (themselves using FORTRAN as the base language) including GPSS, CSL, SIMSCRIPT (Markowitz et al., [1963]) and SIMULA. In the late 60's, revised versions of these languages appeared including GPSS II/III, SIMULA 67 and ECSL (Clementson, 1982). The 1970's was a period of slow development for simulation in which new languages were introduced that permitted the combination of discrete and continuous components in one model. During the fifth stage which spans from the late 1970's to the present day, attention shifted from adding more powerful functions to existing languages to one of providing a more formalised modelling approach which could be used as a basis from improved productivity in code creation. The CASM (Computer Aided Simulation Modelling) project at the LSE as described by Balmer and Paul[1986] is one such example. The CASM concept centered around the use of an Interactive Simulation Program Generator (ISPG) and a suite of PASCAL simulation routines based on systems developed at Lancaster university. The formalism of an Activity Cycle Diagram (ACD) is used as a basis for input into the generator. Also under the umbrella of the CASM project, was work undertaken by Doukidis (See Paul and Doukidis[1986]) on automating the process of model formulation using a Natural Language Understanding System (NLUS). The Pascal simulation routines used in the CASM projects are well documented and tested and are consequently of potential benefit to the research in this thesis. These routines form part of the Extended Lancaster Simulation Environment (eLSE) and are well described by Chew[1986].

The terms 'simulation' and 'modelling' have a widespread and varied usage. Consequently, their meaning in the context of the thesis requires some clarification.

2.2.1 What is Simulation Modelling?

In general terms, 'Modelling' refers to the process of constructing a scaled down version of an existing or proposed real world system. The intention in building a model is to create either a physical replica such as a three dimensional object or to generate an alternative system which does not have the same physical connotations but can nevertheless be used in investigating the properties of the system being modelled. In this second category one can include computer programs used in implementing various types of simulation and mathematical models consisting of series of equations or logical propositions. A prime example is mathematical programming in which a set of linear equations and inequalities are used in creating a model which has no direct equivalence in the real world system being modelled.

Each form of modelling has its strengths and weaknesses. Mathematical or analytic models are powerful with respect to the level of generality of their associated solution techniques. However, such an advantage leads to the converse disadvantage of making it difficult to make model behaviour match that of the real world. In the case of simulation models, where the analogy between the model representation and the real world are that much greater, there is a singular lack of generality, power and elegance as compared to the compact mathematical solution technique. On the other hand, considerable benefit is to be gained by greater faithfulness to detail in that investigation by experimentation is made possible by allowing analogical relationships with the real world to be maintained.

2.2.2 What is the purpose of simulation modelling?

A simulation model is simply a statement of the way in which the various components of a real world system interact to produce a behavioural pattern. The implementation of the model on a computer permits time scales to be reduced to a manageable level and hence permits the program code to be used as a basis for experimentation. Pidd[1992] and McArthur et al[1986] identify a number of reasons for justifying the cost in time and effort of developing a model:

Repeatability: In the case of direct experimentation on a real world system, replication using differing parameters is often either impossible or undesirable. Take a manufacturing plant as an example. Managers and customers alike would be rather unhappy if 'live' experimentation led to a sharp deterioration in delivery lead times.

Danger: Direct experimentation on a real world process can be dangerous. Experimenting, for example, with the operating characteristics of a nuclear power station or aircraft may be unwise.

Time: Specifying the logic of a model and implementing it as program code can take an inordinate amount of time. On the other hand, once implemented, the model can be used to run innumerable experiments on a time scale drastically reduced from that of real time. (e.g., economic systems could not possibly be experimented on directly because of the time factor.)

Inevitability: Some real world systems, such as the solar system cannot be manipulated directly.

Cost: Simulation models are typically expensive to develop given that skilled analysts and programmers are required over a significant period of time. Nevertheless a rash decision implemented as an operating policy on the real world system can turn out to be more costly.

In deciding whether or not to develop a simulation model, the eventual goal(s) have to be identified. As pointed out by Shannon et al.[1985], experimenting with alternative operating policies or procedures is not the only potential use of a model. A simulation model permits the acceptability of the corresponding real world system to be evaluated, either in terms of robustness or performance, and in accordance with a given set of criteria. Sensitivity analysis can be used in identifying the factors which are most significant in affecting system performance. Optimising procedures can be used to fine-tune system performance. An investigation can be made into establishing the functional relationships that exist between one or more parameters in the system. Finally, a model enables transient behaviour such as queue buildups, bottlenecks, and utilisation levels to be identified.

2.2.3 Simulation model development and experimentation.

1. Definition of the problem based on an analysis of the actual or proposed real-world system.

↓

2. Assessment of the feasibility of the simulation, drawing on relevant experience in solution techniques.

↓

3. Identification of objectives and critical system components.

↓

4. Formulation of a conceptual model followed by its representation as a communicative model.

↓

5. Creation of a programmed model.

↓

6. Design of experiments leading to the validation of the model and the presentation of model results. Return to previous stages in developing modified versions of the model.

↓

7. Transfer of the model conclusions to the real world application.

FIGURE 1 STAGES OF THE SIMULATION LIFE CYCLE

In broad terms, the development of a simulation model involves the implementation of the model of the real world system using either a general purpose high level language or simulation specific programming language, followed by an investigation of the model through experimentation.

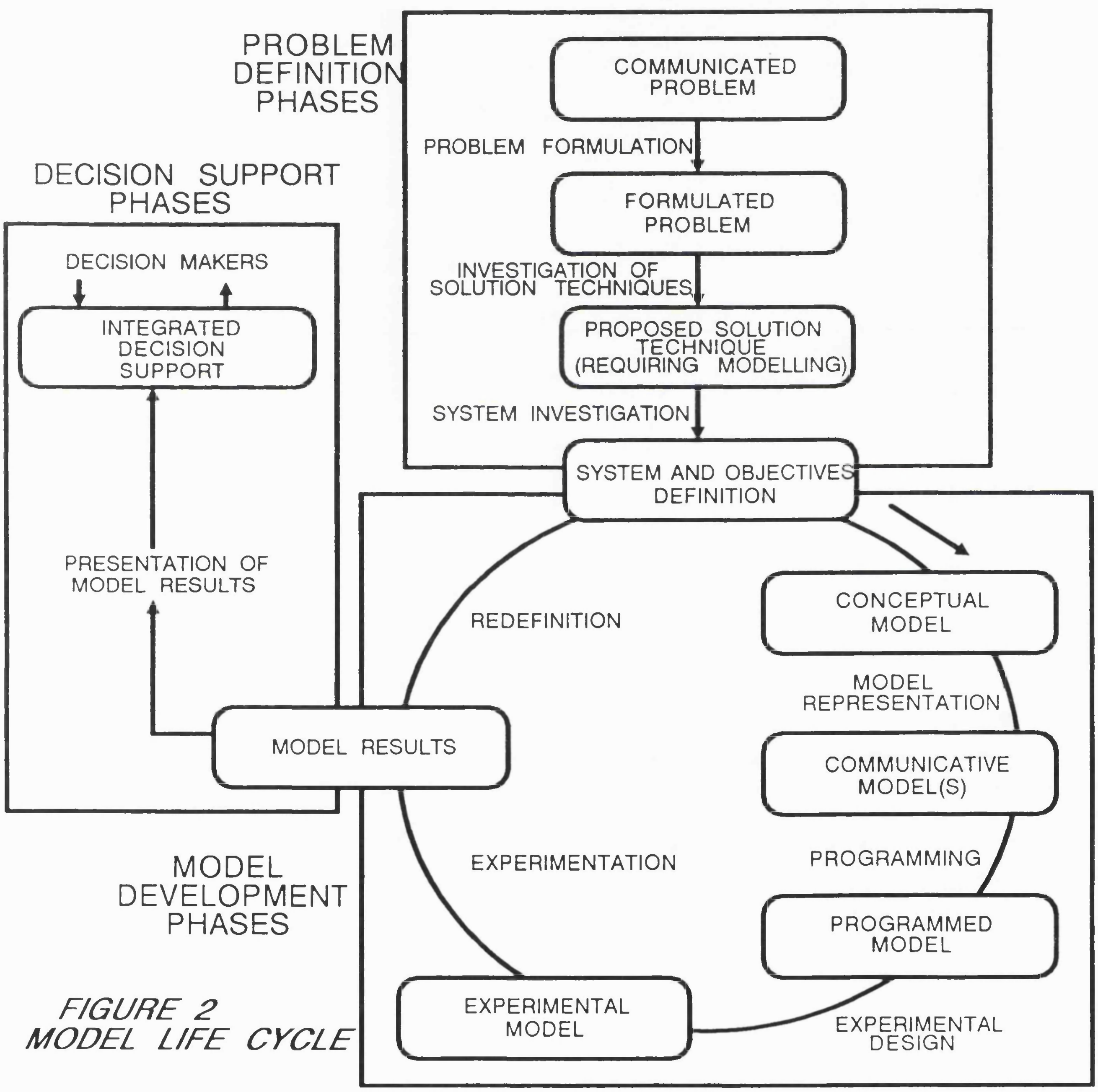


FIGURE 2
MODEL LIFE CYCLE

A proposed simulation model life cycle is defined by Nance[1981] & Balci[1986] and is illustrated in figure 2. The basic stages are listed in figure 1.

As with conventional program development life cycles, the process must be treated as iterative, particularly in the model validation stage, where through display of output, errors or omissions in the logic of the model typically become apparent. Furthermore, experimentation requires a re-analysis of the logic of the model and implementation of such changes through modification of the program code.

2.2.4 Limitations of the simulation modelling approach.

Simulation model formulation and implementation forms part of a challenging and complex process that demands of the modeller considerable analytic skills. The experimentation stage that follows the construction of the model also exacts specialised skills in statistical design and analysis of experiments. Another point noted by Moser[1986] is that the experts needed in interpreting simulation results do not all come from a simulation background and include specialists from the field being investigated. The potential scarcity and cost of such expert advice for output interpretation can nullify the advantages of simulation as a management planning tool. Such limitations linked with the fact that the process of simulation modelling requires long and complex computer programs have lead to the reputation of simulation as a costly and time-consuming process. Pidd[1986] claims that such factors are serious limitations and consequently that 'computer simulation should be regarded as a last resort - to be used if all else fails'. Indeed, many see the primary contribution of simulation to decision support as being limited to areas of high risk strategic decision making in which physical danger or capital investments are major factors.

Simulation modellers face a number of other limitations that cannot easily be overcome and these are typically acknowledged as shortcomings which are offset by the benefits that the model occasions (See Koskossidis & Davies[1987] and Fishman[1973]). Some such limitations can be classified as follows: (The first two have also been discussed in chapter one)

Accuracy: Accurate representation of the real world tends to be a difficult goal to reach, particularly in cases where extensive use is made of simple approximations based on prior observation and sampling rather than modelling actual behaviour. Many systems include the presence of one or more decision makers who typically have considerable influence over the activities that take place. The complexity of decision making tasks sometimes mean that simplification or omission are necessary in creating the model. As O'Keefe and Roach[1987] explain, the difficulties in using present modelling structures for the representation of decision making leads to an inclination on the part of the modeller to limit the level of detail. This limitation in the the level of detail with which decision tasks are represented may have the effect of restricting the scope for experimentation. Another problem sited by the authors is that much critical knowledge can also get lost or misinterpreted during the translation to computer code. It should however be noted that the reverse situation can be just as much of a problem. The gains achieved by an overly detailed model may be totally outweighed by the development overheads incurred and the difficulties that ensue in modifying the model logic. Consequently, a careful balance is required between the level of detail and the investment necessary in achieving the degree of representational accuracy.

Adaptability: As we have seen, the scope for the application of simulation is limited. Other modelling techniques may be more appropriate or there may exist inherent problems in creating a faithful representation of the real world. Given that the simulation model has been created, the modeller may, (at the experimentation stage, and sometimes earlier if systems analysis and design methodologies are not adhered to,) be faced with the need to alter the logic of the model. Meadows[1988] highlights this problem in terms of incomplete or incorrect problem specification which results in a need to make multiple alterations to the code simply to incorporate one modification. McArthur et al.[1986] state that these limitations result from difficulties in structuring knowledge. They also maintain that any structure achieved in the initial model typically becomes lost as more complexity is added or modifications made, with embedded assumptions being 'hidden, scattered, and fragmented throughout the program.'. Such problems are exasperated by the complex interaction of model entities and the difficulties in maintaining consistency in the data that reflects current system status. Such a problem is highlighted in the port application described in chapter five.

Maintainability: Simulation modelling is partially a cyclical process requiring the modeller to switch between experimenting with the model and modifying the code in testing alternatives. The need to repeatedly alter the model imposes intolerable burdens on the structure and maintainability of the code.

Ease of use: As pointed out earlier, experimentation necessitates modification which means that the analyst and programmer have to be involved throughout the duration of the model life-cycle.

Speed: Even at the best of times, long simulation runs are time consuming. Consequently, repeat runs necessary in investigating a range of alternative parameter settings can be a problem. In some cases, simulations run slower than real time potentially eradicating any gain in developing and using the model. (See McArthur et al.[1986])

Validation: According to McArthur et al.[1986], '...there is no assurance that the simulation embeds an accurate or complete model of the dynamic system.'. Consequently, the modeller cannot have total faith in any results obtained from the model and can only attempt to gain an acceptable level of confidence through 'verification' of the computer program and by demonstrating an acceptable correspondence between the output of the model and any actual or historic data. (See Greig[1979], Kheir & Holmes[1978], Mihran[1972], Naylor & Finger[1967], Schlesinger[1974], Schruben[1980], Van Horn[1971] and Koskossidis & Davies[1987]).

Interpretation: Simulation models typically produce a mass of data. If the modelling exercise is to be of any value, the data has to be correctly interpreted which is an error prone and time consuming process. McArthur et al.[1986] give military simulations as an example, emphasising the difficulties in isolating the critical behavioural properties from 'hundreds of pages of numerical output'.

2.2.5 The relationship between simulation and Decision Support Systems.

Decision Support Systems (DSS) are flexible computer based systems that help the decision maker utilise available resources in reaching a specific decision in an unstructured environment such as management and operational control or strategic planning. As stressed by Gray and Borovits[1986], the role of a DSS should not be misunderstood. The intention is to provide support rather than generate specific solutions which the user accepts as a final decision.

Simulation modelling, by providing an insight into the functionality of the real world system, is itself fundamentally a tool for the support of decision-making. Where simulation modelling differs from the concept of a decision support system, is in the level of support provided to the user. According to Nathan and Sokol[1986], simulation neither relates the simulation results to a manager's multiple and conflicting objectives, nor does it directly assist the manager in identifying the best solution. Simulation models have a number of other limitations in the context of decision support; Large amounts of output are produced with no direct means for comparing the effects of changes to the model parameters in different runs. Furthermore, an individual, when experimenting with the model, needs to be guided rather than left to the slow process of 'trial and error'. This problem is further compounded by the fact that the analysis of the output of a stochastic simulation requires a deal of statistical expertise and cannot sensibly be left to a busy manager faced with an urgent decision. Even if such statistical analysis could be reduced to a simple routine and the whole embedded within an optimising algorithm, the execution of the multiple replications of each of the alternative decision scenarios required by such a process may pose intolerable computational burdens for an on-line decision support system.

Taking an alternative viewpoint, simulation modelling can be seen to make significant contributions to decision support systems. For example, The actual process of developing a simulation model may occasion within the user an enhanced appreciation of the operation of the system modelled. This may, in itself prove useful in supporting decision making or may contribute indirectly to the process of creating a DSS. Another possibility is that the model produced could be used in a formal series of experiments which potentially culminate in a rule or set of rules that are then used as part of the DSS. For instance, a regression model could be fitted to the simulation output which then adequately summarises the effects of changes in certain input parameters. The changes to the input parameters could represent alternative decisions on the operation and management of the real world system and consequently, the regression equation could be incorporated in the DSS which then requires no further reference back to the simulation model. (See Nathan and Sokol[1986])

Another common approach has been to embed the model within the DSS thus allowing the simulation to play a direct role in decision support. A system developed by Basset and Kochhar[1985], provides data analysis and report generation routines but does not provide the user with any degree of flexibility and the problems of using simulation for decision support highlighted in the previous paragraphs remain unresolved. Other writers such as Moser[1986] take a similar approach but rely on a rule-based expert system for the interpretation of the simulation output. This expert system is developed in parallel with the simulation and embodies the knowledge of both simulation analyst and domain experts used in the interpretation of output. Another system formerly known as KBS and now named Simulation Craft takes a far more ambitious approach (McRoberts et al.[1986], Reddy et al.[1986], Reddy[1987], and Sathi et al.[1986]). It is proposed that Simulation Craft be capable of identifying appropriate sets of scenarios, automatically generating a number of experiments such that the stated 'goal' be attained and producing a report explaining the scenario selected. Such a system offers a functionality which, if fully realised, reserves for simulation a place within the realm of on-line DSS.

The next section will provide an overview to the area of AI and in particular, expert systems, prior to investigating the research that has been undertaken in combining characteristics of simulation and AI.

2.3 ARTIFICIAL INTELLIGENCE.

The roots of Artificial Intelligence are widely accepted as dating back to 1950 when Turing[1950] wrote his speculative paper on computer machinery and intelligence. (In comparison, the first commercial computer, the IBM 705, appeared in 1954 and the first programming language, FORTRAN, in 1957). In

1956, a conference at Dartmouth college on symbolic computation paved the way for the development of practical applications. However, it was not until the 1970's that the concept of AI was to find acceptance outside research environments. Unfortunately, interest dwindled because the AI applications were too slow coupled with high development costs and small practical returns (Harmon and King[1985]). It was not until the 1980's that AI was finally to gain acceptance, and not so much because of any significant theoretical advances, but because developments in chip technology led to the introduction of a new generation of substantially more powerful computers at relatively lower costs.

AI is concerned with how humans 'acquire, organize, and use knowledge' (Shannon et al.[1985]). The constituent areas of AI are not clearly defined but broadly fall into three classes. Natural language processing, robotics, and knowledge based systems.

Natural Language Processing (NLP) is primarily concerned with the development of computer applications that can read documents, speak, and recognise spoken words (speech recognition). The interest in NLP is spearheaded by a need to provide a more powerful means of communication between man and computer, coupled with the commercial availability in recent years of, text scanners, speech synthesisers and speech recognition equipment. (See Winograd[1972] for a more detailed coverage of NLP).

Robotics is concerned with how robots can be given tactile and visual senses. Dramatic advances have been made in recent years, resulting in wide usage of such technologies in industry, primarily in the context of Automated Guided Vehicles (AGVs), image recognition (e.g fingerprint identification), and machine guidance (e.g. welding and cutting in the car manufacturing industry). The scope for the use of robotics is substantial as businesses cut overheads in striving to remain competitive. (See Pratt[1978] and Brooks et al.[1979] for a more detailed discussion of the topic).

Knowledge based systems include Expert Systems and Neural Networks, a new area of research mainly dedicated to machine learning. Expert Systems (ESs) are concerned with the automation of mental tasks that are normally undertaken by an expert in a specific application area. Expert systems differ significantly from other AI applications, namely NLP and robotics, in that the underlying goal is not that of gaining an insight into how human experts reach a given conclusion, but rather, that of devising methods by which such conclusions may effectively be duplicated (Shannon et al.[1985]). The research presented in this thesis is primarily concerned with the contributions that expert systems can make to simulation, and consequently, the following sections will focus exclusively on ES theories.

2.3.1 What are Expert systems and how do they work?

According to Feigenbaum[1982], expert systems are intelligent computer programs that use '...knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution.' Expert systems differ from conventional problem solving techniques both in terms of the development process and architecture of the implemented end product. The procedural approach used in conventional high level languages is abandoned in favour of an architecture which is typically based on the use of three distinct modules that represent the knowledge of the system. The three components are:

A database (or equivalent) for the storage of data corresponding to the 'declarative knowledge' to be used by the ES, and run-time data representing the current status of the system. (declarative knowledge is data, specified before the start of the inference process).

A knowledge-base which encapsulates the facts and rules that embody the expert's domain knowledge.

An inference engine that consists of deductive strategies that define the problem solving approach to be used. The inference engine analyses available facts and rules and attempts to draw conclusions which get added to the database or are used to modify current database entries. The inference engine is further responsible for instigating order in the pattern of inquiry.

1.Knowledge representation: Expert system shells which are high level tools for the creation of expert systems typically provide a language construct, data structures, a generalised inference engine, and a user-friendly interface. There exists several forms of representation for the facts and knowledge stored in the database and knowledge-base. The most common of these are: Semantic networks, frames, object-attribute-value triplets, and predicate calculus. Other forms of representation including program code, rules, conditional probabilities, and first-order logic which are used almost exclusively in representing domain specific knowledge in the knowledge-base.

Semantic networks are one of the oldest and most general representation schemes for declarative knowledge (Harmon and King[1985]). Objects to be represented are symbolised by nodes and relationships denoted by arcs that link the objects together. The advantage of such a representation is the clear image that can be obtained as to relationships between objects through graphical representation compared to lines of code in a classical program. Semantic networks are flexible inasmuch as new nodes and arcs can be added as needed and have the benefit of permitting objects to inherit the attribute values of other objects through the creation of additional arcs.

Frames are a form of representation for objects which contains slots for the storage of facts about the object. The slots may contain values or pointers. A pointer may point to another frame or alternatively to a procedure or set of rules that return a value. Consequently, frames are capable of both procedural and declarative representational forms. As with semantic networks, frames can inherit the attribute values of other objects. (See Alty and Coombs[1984])

Object-attribute-value triplets are similar in concept to semantic networks. The arcs used in semantic networks to symbolise relationships are simplified by only allowing two kinds of relationships. Namely, "is-a" and "has-a" arcs. O-A-V triplets were used in the MYCIN medical diagnosis expert system (see Buchanan and Shortliffe[1984]).

Predicate calculus is a simple language for the definition of objects and facts (predicates) relating to these objects. The format of statements in predicate calculus consist of a fact followed by one or more object names between parentheses. For example, "Is-AssemblyMachine(Mach_A)" is equivalent to the statement that "machine A is a machine for the assembly process". Such an assertion can either be TRUE or FALSE. Predicate calculus has the advantage of being fairly English like and yet has a simple and limited syntax. (See Alty and Coombs[1984])

Program code is often used in conjunction with other knowledge representation structures in defining the domain specific knowledge. A procedure may be called when a given set of conditions are satisfied. A number of expert system shells provide facilities for interfacing to conventional high level languages though data sharing is often impossible or awkward to use.

Rules (production rules) typically have an IF-THEN type structure consisting of a premise and conclusion which can be grouped together using logical operators. The premise is used to check the current state and if satisfied, results in a modification of the current state through activation of the statements declared in the conclusion. Some production rule languages provide facilities for conditional probabilities which permit rules to conclude results that only have a certain probability of being correct given that the premise has been satisfied. (see Buchanan and Shortliffe[1984] for a description of reasoning about uncertainty in

MYCIN).

First-order logic, or more specifically, Horn clauses can be used in defining knowledge in either a declarative or procedural sense (See Futo[1985], Bullers & Schultz[1986], Cleary et al.[1985] and Adelsberger[1984]). For example, B :- A1,..,An. can be interpreted as a logical statement that says that B is true if A1 to An are true. Alternatively, in the procedural sense, the statement can be interpreted as being that the problem of evaluating B is reduced to the sub-problem of evaluating A1 to An. A more detailed evaluation of first-order predicate logic is reserved for a later section in a discussion of the facilities provided by Prolog.

2.Inference & control strategies: The inference engine is the part of the expert system that embodies the strategies that are used to draw inferences from the facts and rules declared in the database and knowledge-base, and that controls the reasoning process. The inference engine also acts as an interface between the end-user and the stored knowledge, effectively conducting a consultation whilst drawing on the knowledge to provide solutions.

At the simplest level, the inference strategies used by expert systems simply consists of statements that say that if the premise of a rule is true, that the conclusions can then be accepted. Modification of this principle also exist such that a basic statement of the type IF A THEN B can also be taken as meaning that if A is NOT true then that B cannot also be true. Another possibility is that if the value of B can be evaluated (but not that of A), that we can then derive the value of A without the need to explicitly code this fact using another rule. As mentioned in the section on rules, probabilities can also be associated with statements that reflect the uncertainty of the validity of given information.

At the control level, the inference engine must organise the steps taken in solving a problem. The inference engine is also responsible for the following tasks:

1. Selecting a position from which the reasoning process can begin.
2. Resolving conflicts in logic between rules.
3. Choosing, a rule from a set of rules that can all be evaluated.
4. Interrupting the inference process in order to obtain missing information from the operator.

The two most common control strategies are forward chaining and backward chaining, the use of which will depend on the problem domain. Furthermore, forward and backward chaining can either be carried out using a depth-first or breadth-first searching strategy.

A forward chaining or data-driven strategy is usually employed when the desired goal is not initially known. This is typically the case in monitoring systems in which there is no goal to commence the inference from. In forward chaining, the start conditions consist of the current entries in the database and the inference proceeds by identifying those rules that have premises that can be satisfied. The action part of the statements are executed resulting in further facts being added to the database. The process is then repeated until either a desired state is reached or until no remaining premises can be satisfied. The difficulty with the forward chaining strategy is that at each step in the cycle, a choice has to be made between a number of rules that have premises that are satisfied. As the number of such rules increases, a noticeable deterioration in performance is felt consequent to the increased complexity of the selection process.

A backward chaining or goal-directed strategy is used in circumstances where the desired end goal is known. The goal is evaluated by searching for a rule (or rules) that has an action that satisfies the premise of the goal. This rule is then defined as a sub-goal and the process repeated until the premise of the original goal is satisfied or until no more sub-goals can be identified. If the search strategy is 'irrevocable' and the goal is unresolved, the inference engine can proceed no further (see Shannon et al.[1985]). Alternative paths through the solution space can only be attempted by re-commencing the inference process. If the search strategy is 'tentative', the inference engine can backtrack to an earlier sub-goal, select a new rule, and again endeavour to find a solution.

In Depth-first searching, priority is given to producing sub-goals. Hence, alternative paths through the solution space are only considered once a particular path reaches a dead-end. If the expert system interrogates the operator for input, the feeling given is one of a search which results in questions of ever-greater detail.

In breadth-first searching using a backward chaining inference strategy, consideration is given first to alternative sub-goals. In other words, all paths that could lead to the solution are investigated simultaneously. The efficiency of breadth-first searching is dependent on how quickly a rule premise can be found that satisfies the goal. Breadth-first searching tends to be unpopular in systems that require substantial user interaction consequent to the operator feeling uneasy about having to answer questions that seem to be ordered at random.

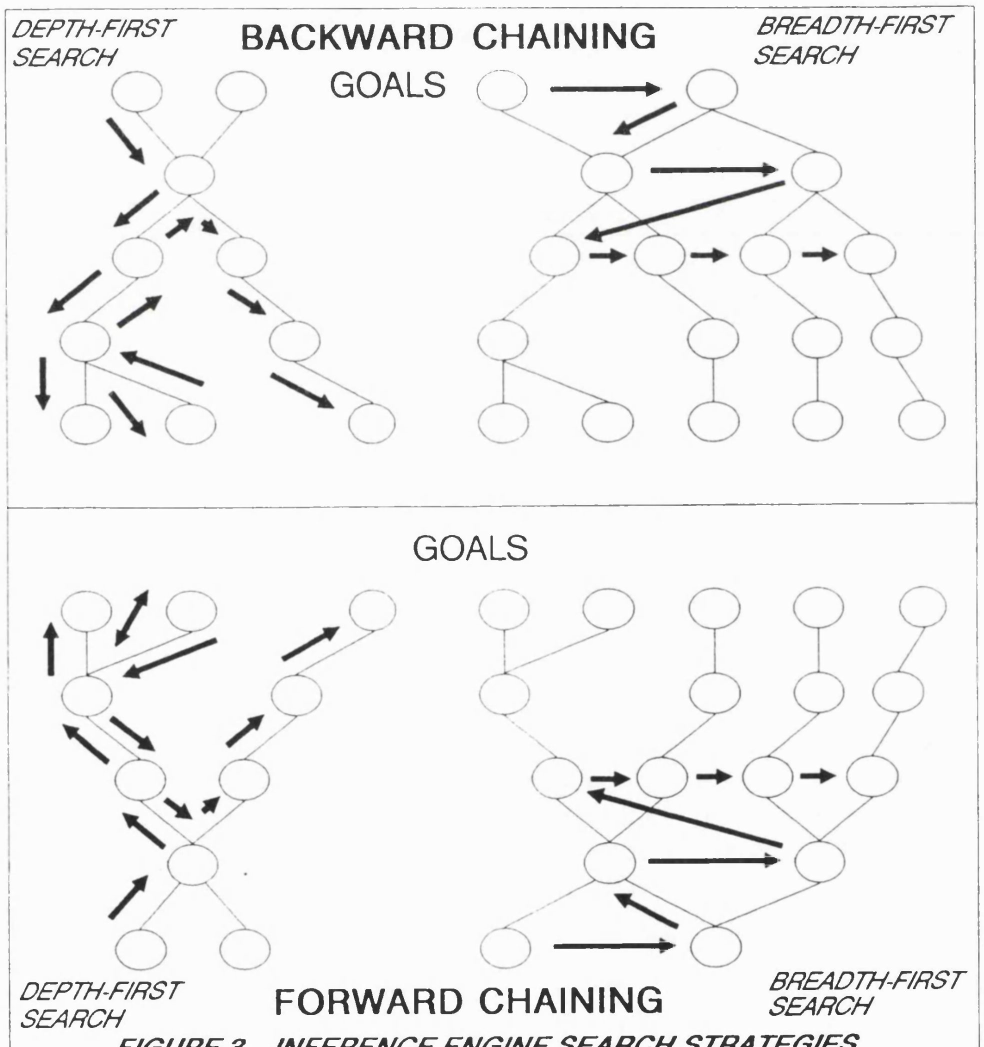


FIGURE 3 INFERENCE ENGINE SEARCH STRATEGIES

2.3.2 What is the purpose of an expert system?

The principle underlining the expert system approach, is to enable the representation of the knowledge of one or more experts within a specific domain. For example, in Fox and Smith[1984] the pertinent expert knowledge is concerned with the scheduling of jobs within a machine shop. This knowledge-base is searched to provide answers to questions such as which jobs should be given priority if a goal of ensuring that contractual agreements on delivery dates has to be met.

In many cases, the knowledge represented in the developed system relates to some complicated decision-making process but cannot be described as an expert's knowledge. Consequently, expert systems can be used in a method akin to conventional programming in situations where the inference strategy and incremental development process of a knowledge-base are deemed advantageous.

Expert system shells often attempt to provide facilities that in some ways resemble the approach taken by human experts. For example, experts often need to consult others in solving problems and consequently the expert system may incorporate facilities for interrogating the operator. The user can typically skip questions or associate an uncertainty factor with an answer. The expert system can usually explain a line of reasoning and justify conclusions, though the facility is limited in that the output obtained is typically a trace of the inference through the knowledge-base.

The most common use of expert systems is as advisory systems in which some form of interactive consultation takes place. Two of the best known examples are MYCIN (Shortliffe[1976] and Buchanan & Shortliffe[1984]) for medical diagnosis and PROSPECTOR (Duda et al.[1979]) for the analysis of geological data. Another common use for expert systems is for training and educational purposes, either through modification of an existing ES (Buchanan & Shortliffe[1984]), or by using an approach that 'customises' the teaching session according to past attainment. Research is also being carried out into the use of expert systems as integral modules in software for on-line decision making for manufacturing process control (Brown et al.[1985]) and as intelligent front-ends (Muetzelfeldt et al.[1985]).

2.3.3 Limitations of expert systems.

Many researchers (and particularly those not working in AI) have cast doubt on the effectiveness of the expert system approach whereas others have stopped just short of heralding its discovery as the dawn to a new era. The main argument put forward by the sceptics is that the ES approach is to produce a system which externally manifests the behaviour of the relevant expert but that internally, uses an unnatural format for the representation of the experts knowledge and uses an inference strategy that is a crude simplification of the way the expert thinks. The knowledge of an expert in a diagnostic domain is believed to consist of both a mental model of the problem and rules of thumb which are used to guide the diagnostics process. This mental model is flexible in that it is adaptable to similar problem domains by permitting analogical reasoning. Alty[1985] points to the deficiency of using IF-THEN type rules (production rules) by exemplifying the difficulties in transferring knowledge represented in this format to other applications. Production rules are an ideal representation for the rules of thumb used by the expert, but otherwise necessitate a considerable amount of domain knowledge to be discarded at the expense of the addition of extraneous computational knowledge..

The use of production rules leads to a tendency to expand the knowledge-base incrementally as rules are elicited from the expert. This can lead to an scattering of rules which inevitably results in a system which is either incomplete, ambiguous or inconsistent. Poor performance of the ES results from difficulties in maintaining order in the knowledge-base which would otherwise benefit the relatively simplistic search and pattern matching procedures of the inference engine (Muller[1986]).

Maintenance of a knowledge-base is fraught with difficulties resulting from an inability to manually trace through the logic of the system consequent to the scattering of the production rules and the lack of an explicit definition of the expert systems inference strategy. Most expert systems consequently incorporate an 'explanation' facility that lists the rules that are activated at each step in the inference process.

Most expert systems are poor at incorporating algorithmic approaches to supplement the rule-based reasoning and many have no facilities for executing procedural code.

Expert systems handle decision making as an instantaneous process, whereas time may be a critical factor. An ES controlling a production line using on-line data may for example have to carry out forward projections in reaching a decision. The incorporation of time into the inference mechanism blurs the distinction between expert systems and simulation models. (Miller[1986])

2.4 AI AND SIMULATION MODELLING - MUTUAL SUPPORT.

Previous sections have identified the general characteristics of both simulation and expert systems, as well as the shortcomings and benefits of each approach. The similarities between simulation and expert systems are now considered with emphasis being placed on the possibility of adapting an expert system to carry out the role of a simulation model (and vice-versa). The possible ways of integrating simulation and AI techniques are then considered.

2.4.1 Expert systems and simulation - Is there a difference?

Operational similarities between simulation and expert systems, have been noted by several writers. As pointed out by Shaw and Gaines [1986], an expert system can be considered as the simulation of the external manifestations of the knowledge processes of a person. A more subtle relationship also exists in that simulation and expert systems are both aids to individuals in coping with the real world. Simulation does not provide direct advice but, through the use of a model, permits experiments to be carried out giving the user a greater understanding of the system being modelled and permitting the investigation of ideas before implementation in the real world. Similarly, expert systems can be seen, through advice giving and explanation facilities, as providing a means of gaining an increased appreciation of the criteria applied in reaching a decision.

Advice provided by the ES may then be applied in the real world.

Similarities between simulation and expert systems have also been identified at a methodological level. For instance, Doukidis[1987] argues that a "...three-phase simulation system can be seen as a production system", his reasoning being that the three essential components are present: Data memory, production model, and inference engine. In discrete event simulation, model execution is effected through a three-phase executive which performs a time-advance in the A phase, executes all current time-dependent events in the B phase and examines and executes where appropriate all state-dependent events in the C phase (Tocher[1962]). The executive can be compared to a forward chaining inference engine, which, at each time-advance, scans the state-dependent C events (the production rules) in search of routines that can be activated. The definition of the model logic, separate from the executive controlling model execution, gives three-phase simulation some of the characteristics of a declarative language. A diagrammatic representation of the three-phase approach is shown in figure 4.

Though the general structure of a discrete event simulation model and an expert system are quite clearly similar, the strategies applied during the inference processes are significantly different. Discrete event simulation is primarily concerned with time handling and the representation of the activities that constitute the model. Little consideration is given to the representation of decision making. Conversely, expert systems rely on a detailed description of decision rules with no consideration being given to the effects of time. The expert system inference strategy is a general one that permits the application of expert systems to a variety of problems. Hence, a simulation model can be specified using a declarative expert system approach in which the time handling capability is defined in terms of production rules. Although feasible, expert system production rules are not an ideal medium for the representation of simulation entities and activities and the approach is consequently of no significant benefit. An alternative approach is to adapt the strategy used by the inference engine to represent discrete advances in time by maintaining a diary of scheduled events. The necessary alterations are significant and prevent thereafter the use of the inference engine in its traditional role. The approach has been investigated by Robertson[1986] and is described in more detail in section 2.5.4.

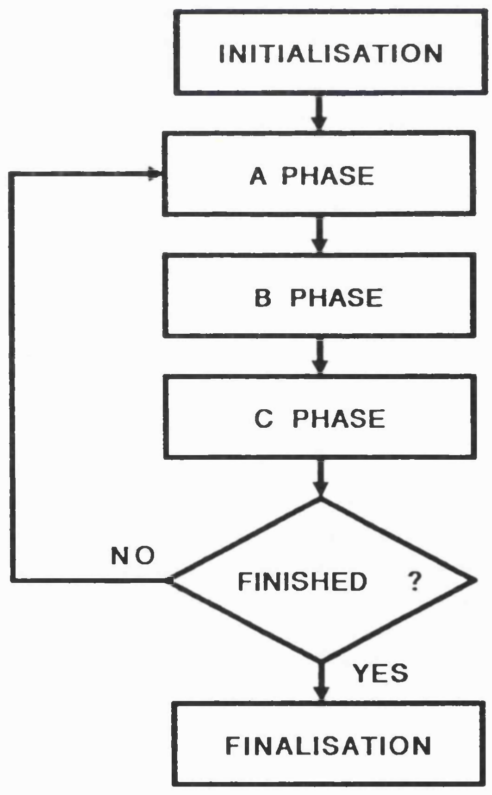


FIGURE 4: THREE PHASE SIMULATION

2.4.2 Simulation and expert systems - Complementary techniques.

Many applications have been developed that in some way make use of both a simulation model and an expert system. Such applications have evolved from a realisation that the strengths of simulation complement the weaknesses of expert systems and vice-versa. The potential for interaction between both technologies has been noted by many writers (O'Keefe et al.[1986], Helman & Bahuguna[1986], Flitman & Hurrion[1987], Hill & Roberts[1987], and Shannon et al.[1985]).

Researchers including Fox & Smith[1984] and Brown et al.[1985] are investigating the use of expert systems as core elements in decision support systems. In such systems, simulation can be used to the benefit of AI, by using a model for the generation of test data which would normally be accessible to the DSS. Thus simulation is being used to reduce the effect of one of the shortcomings of ES methodologies, namely validation of the knowledge-base. Brown et al. use the ES for the detection of tool wear and fixture faults in a hypothetical drilling process. The viability of the approach is being investigated using simulation modelling in re-creating the operational environment. Stewart and Surgenor[1987] follow a similar principle by using a simulation model for the validation of a prototype ES for fault diagnosis in a production plant. The simulation model and expert system are implemented on separate microcomputers with simulation output being transmitted for diagnosis to the expert system. Consequently, the ES can be validated using a wide range of realistic data reflecting potentially rare occurrences such as multiple simultaneous faults.

Expert systems can also be of benefit to simulation modellers. One of the shortcomings of simulation mentioned in section 2.2.4 is the necessity for considerable expertise in producing the model and analysing the generated output. Advisory expert systems that provide support to the user by embodying the knowledge of experienced simulation modellers are currently being considered by several researchers. Doukidis and Paul [1991] describe SIPDES, a system which helps users to discover the location of compilation errors occurring within their simulation program and proposes possible solutions. Similarly, the experimentation and analysis phases of simulation modelling are being supported by automatic systems such as that embodied in the 'model execution' and 'model analysis' modules of Simulation Craft (Sathi et al.[1986]). The model execution expert is primarily responsible for determining the necessary experiments and the corresponding number of runs that are required. The model analysis expert is claimed to evaluate experiments, generate alternatives, and provide explanation facilities using statistical routines.

Such mutual support activities are clearly beneficial and do not necessitate any direct interaction between expert system and simulation model other than for the sharing of data. Another area for mutual co-operation is in the marriage of ES and simulation techniques in providing a simulation environment that permits the modelling of intelligent behaviour, the handling of events over time, and the representation of algorithmic components of the model.

For instance, in the modelling of a manufacturing facility, the decision-making activities of employees, whether machine operators or top-level management, can be considered as a particular form of expertise. This expertise may be represented in the form of rules which may be clearly-stated instructions, rules of plausible reasoning, or rules of thumb (heuristics). The knowledge of employees is further supplemented by "facts" which may have been acquired through job experience and data which may be publicly available.

The basic functions and performance of machines including durations of operations and the basic processing sequences of product are well described by the conventional data structures and are well handled in conventional procedural languages usually used in simulation. Any material requirements planning functions depending on orders outstanding and current cost data can also be well accommodated within a procedural framework.

In contrast, decision tasks of any significant complexity may be difficult to integrate with the discrete event model. This is because decision tasks are often broken down into a significant number of related rules which are difficult to define in the sequential order required by a procedural language. In forcing a procedural context, rules have to be repeated within the code and the associated between such rules formalised through the use of logical operators. In contrast, the declarative programming approach as used in expert systems, permits the formalisation of decision tasks through the definition of component rules but without any requirement for order and without the need to link the rules through logical operators. Instead, the expert system inference engine embodies an inference strategy which is used to scan the defined rules in an attempt to satisfy the conditional statements.

Various methods have been considered in combining the functionality of ESs and simulation. Some researchers, and often those with a strong background in artificial intelligence have opted for using AI languages, usually LISP or PROLOG. In the United States, LISP is the main language used in AI and so there is a natural inclination towards its use in this context. Most LISP based simulation environments operate according to the object oriented programming paradigm. In Europe and Japan, government sponsored research has given the PROLOG approach the leading edge. In some cases, modified versions of standard PROLOG have been used that are tailored to simulation. Another approach, though usually discussed rather than attempted, is to interface a simulation model with an expert system. The difficulties with this approach consist of implementing an adequate form of communication between functionally incompatible software. An approach which has been used in overcoming this problem is to implement the simulation model and expert system on separate computers and achieve data sharing through a generalised communication protocol (see section 2.5.5).

2.5 WIDER ASPECTS OF AI SUPPORT OF SIMULATION MODELLING.

In recent years, much research has been carried out on improving the performance of simulation models by confronting the problems highlighted in section 2.2.4. Such research, coupled with improvements in hardware and falling prices in the area of personal computers and workstations has resulted in the possibility of applying simulation modelling techniques to a far wider range of applications.

The level of capital investment necessary in undertaking a simulation study is now less of an issue, particularly in the context of microcomputer based systems where the presence of easily accessible, colour graphics has promoted the growth of windowing environments and iconic displays. The overall effect has been that researchers have focused their interest on developing tools that enable the relatively inexperienced simulation modeller to define and develop models, devise experiments and then analyse simulation output without the need to call on the resources of more experienced practitioners.

2.5.1 Simulation program generators.

Researchers have invested considerable time and effort in the development of simulation program generators with a view to reducing the necessary time span in the model creation stage of the simulation modelling life-cycle (see Clementson[1982]). A second consideration has been to attempt to devise flexible and user-friendly systems that guide the inexperienced user through a model specification process.

Sathi et al.[1986] place most emphasis on the second consideration and use an expert system that encapsulates the knowledge of simulation experts for an interactive model specification session based on the use of graphics for the description of model components. The ES is also responsible for consistency and completeness checks. Shannon[1986] describes a hypothetical system which is similarly based on the use of icons for model specification and templates for the definition of the actions relating to components.

The use of graphical depictions as a means of formalising the behaviour of a system is a long standing approach to modelling (See Clementson[1978], Matthewson[1975], Gordon[1981], and Zeigler[1976]). Such an approach has the benefit of providing a simple vehicle for discussion between client and analyst and permits the detection of potential logic errors. The main limitation associated with graphical depictions is the difficulty in representing complex real-world systems in which the paths between queues and activities are numerous and often ambiguous. Furthermore, graphical representations omit all references to decision making including conditional branching and batch processing of queue entities. Such problems restrict the value of using graphical model representations as input to program generators as only the simplest of modelling tasks can be dealt with.

A few researchers including Doukidis[1987] take a different approach and rely on a tentative method based on techniques derived from Natural Language Understanding Systems (NLUS). The client and analyst are expected to go through the consultation session together, the end product being a logic model which can in turn be used as input to a program generator.

In Overstreet & Nance[1985] and Balci & Nance[1987], the prototype of a discrete-event Simulation Model Development Environment (SMDE) is described. SMDE includes a model specification and documentation generator as well as a model analyser. The model generator is used in creating a formal model specification which is domain independent and can later be converted into executable code. The advantage with such an approach is that errors detected in the model specification are far easier to correct that errors in source code because of a lack of any stringent syntactic and semantic elements.

Furthermore, the model specification is defined in terms of a simple language referred to as the Conditional Specification (CS). According to Nance and Overstreet, CS strikes a balance between 'descriptive generality and an instructive formalism' which permits the analyst to further develop and test the model before generating the source code.

2.5.2 Model verification and validation.

The current trend in creating development environments has naturally led to research into ways of automating the process of model verification and validation.

Verification is the process of debugging the simulation code and checking that the model operates as intended (See Koskossidis & Davies[1987]). The Simulation Model Development Environment (SMDE) as described in the previous section includes a model analyzer that diagnoses the model specification created by the model generator. The intention is to help identify mistakes, in particular conceptual and descriptive errors, to suggest alternative model configurations that may prove to be more efficient, and to provide general guidance during the modelling effort. The approach taken in SIMULATION CRAFT (Sathi et al.[1986]) is similar though this time, the embedded model building ES is responsible for consistency and completeness checks during the graphical model input process. Other research projects have also used the expert system approach to model verification. SIPDES (Doukidis[1987]) and TIM (Hill & Roberts[1987]) rely on an interactive session with the user in identifying the potential source of compilation and run-time errors.

The generation of execution errors during the model building and experimentation processes are a considerable help to model verification and tend to form the basis for the diagnosis processes in simulation support software. In contrast, the validation of a model is a complex process, necessitating from the analyst considerable skill and experience. The formalisation of such knowledge in the development of an expert system is rendered impracticable by the problem-dependent nature of the validation process (Van Horn[1971]).

Validation consists of ensuring that a model is a realistic representation of the real world and that results obtained and conclusions drawn from experiments can safely be applied to the real world. According to Van Horn[1971], validation is "the process of building an acceptable level of confidence that an inference about a simulated process is a correct or valid inference for the actual process.". Van Horn also claims that a simulation model can seldom, if ever, be proved to be a "true" representation of the real process. The problem of validating a model is compounded by concealed and questionable assumptions that are embedded in the code and result in a tendency to treat the model as a 'black box' that transforms inputs into outputs.

Consequent to model complexities, the validation process only partially relies on an in-depth perusal of the internal model representations. Reliance is instead placed on the use of historical data in comparing model output with observations from the real world. Some researchers have taken the view that involvement of the client in the modelling and testing stages of the simulation life cycle is of considerable help to the validation process. Talavage[1978] describes the development of one such model for which only a small amount of historical data was available. Confidence in the validity of the model was nevertheless attained by involving the client in the analysis of the models behaviour compared to that observed in the real world.

Improvements in graphics and the increased used of iconic displays has led to the growth of Visual Interactive Simulation (VIS) modelling in which changes in the state of the model through time are represented in pictorial form during the simulation runs (Crookes & Valentine[1982], Hurrion[1978], Vujosevic[1990]). The use of animated displays eases the problem of client involvement in the model validation process and improves management confidence in the modelling analogy.

Simulation models do not embody the true complexities of human decision making with the consequent need for simplifications that complicate the validation process. VIS can be seen to represent some response to this problem by permitting the user to intervene in response to observed model behaviour, alter characteristics of the model, and then continue the run with the modified model. By appropriate interventions, decision mechanisms of arbitrary complexity may be achieved. However, this achievement is at the cost of not being able to secure the benefits of replication and statistical analysis of performance.

2.5.3 Intelligent front-ends.

The complexity of certain software solutions, which includes simulation environments, has led to research into ways of using expert systems for ease of communication and information presentation. A technically complex system is wasted unless the user can have confidence in its operations. Such confidence can only be attained if the operator finds the system comprehensible and usable.

An Intelligent Front-End (IFE) is an interface that sits between the software package and the operator and shields the user from complex application specific operational tasks. Through consultation with the user, the IFE generates the necessary instructions to operate the program.

In the context of comprehensive and unavoidably complex simulation environments, Intelligent front-ends provide the potential of maximising on the productive use of the model. The flexible nature of expert systems permits the customisation of the environment, and in particular the dialogue management, to suit the requirements and level of expertise of individual users.

Some researchers have extended the role of the intelligent front-end to encapsulate both the generation of simulation code and the analysis of simulation results. Such facilities go beyond the idea of an IFE as an interface for dialogue handling and make of the front-end an integrated part of the simulation package. Simulation Craft (Sathi et al.[1986]) is probably the most advanced system so far produced and is primarily intended for use in manufacturing domains. Simulation Craft attempts somewhat ambitiously to automate all stages of the modelling process and consequently has three embedded expert systems for model building, model execution, and model analysis respectively. KIPS (Knowledge based Interface to Process Simulation) is a conceptually similar system, implemented on a LISP workstation that acts as a front-end to a mainframe simulation program used in petrochemical process plants. The KIPS prototype is principally a program generator based on the use of a graphical flowsheet editor for input and a so far partially completed knowledge-base for user interaction and guidance (Fjellheim[1985]).

Other researchers have concentrated their efforts on two aspects of intelligent front-end design: (1) The provision of features such as natural language dialogue handling. (2) The formalisation and integration of some of the features of the simulation package into a knowledge-base such that some of the decision making tasks can be taken by the IFE rather than by the user. ECO is an example of such a system and is essentially an intelligent front-end designed to help ecologists construct and experiment with simulation models of ecological systems. The ECO dialogue handler is designed to accept both prompted input and unprompted natural language input though problems with the latter are identified by Muetzelfeldt et al.[1985]. The author also highlights the difficulties in generating models of relatively simple systems that require list or tree-like data structures. The generated code, in this case FORTRAN, needs to be able to iterate over n-dimensional arrays and handle pointer types.

2.5.4 AI languages & tools in simulation.

The similarities between techniques used in simulation modelling and expert systems have been highlighted by several authors including O'Keefe[1986] and Doukidis[1987]. The commonalties are sufficiently great to have led to attempts, using a variety of techniques, to develop simulation models entirely within an AI environment and doing away with the classical simulation methods. Such systems, typically referred to by those working in

AI as 'Knowledge Based Simulation' (KBS) models, are implemented without the direct use of sequential processing techniques. Models developed using the KBS approach differ significantly from models developed using conventional methods. The principal difference lies in the use of rules in representing knowledge. Such knowledge can be classified into two broad categories: heuristics that govern decision making and knowledge that acts as a representation of physical processes and their interplay. KBS models typically provide no facility for separately defining these forms of knowledge. Nevertheless, the control/inference component of the model is represented as a separate and distinct entity from the data component permitting either to be altered independently from the other (Lavery[1986]).

Other peculiarities of Knowledge Based Systems are specific to given modelling approaches. For instance, some researchers have chosen to represent the time-flow mechanism through alteration of the languages inference strategy whilst others have added rules to the knowledge-base to achieve the same ends. Another approach adopted by some researchers, known as goal directed simulation, is based on taking a different view of model representation (Prakash and Shannon[1989]). Use is made of the goal directed inference strategy common in most expert systems in guiding the simulation process. Model process cycles are then represented as goals, the achievement of which necessitate the achievement of sub-goals. For example, in the case of the port model (Chapter five), one could represent a ship crane as having a top level goal of depositing a container on the ship deck. A loaded ship crane and the presence of a ship with spare capacity in the berth are necessary sub-goals. In turn, a loaded ship crane will have necessitated an idle crane, a driver, and an export container. Thus, each object in the system can be specified in terms of series of goals and related sub-goals rather than say, events and activities as in discrete event simulation.

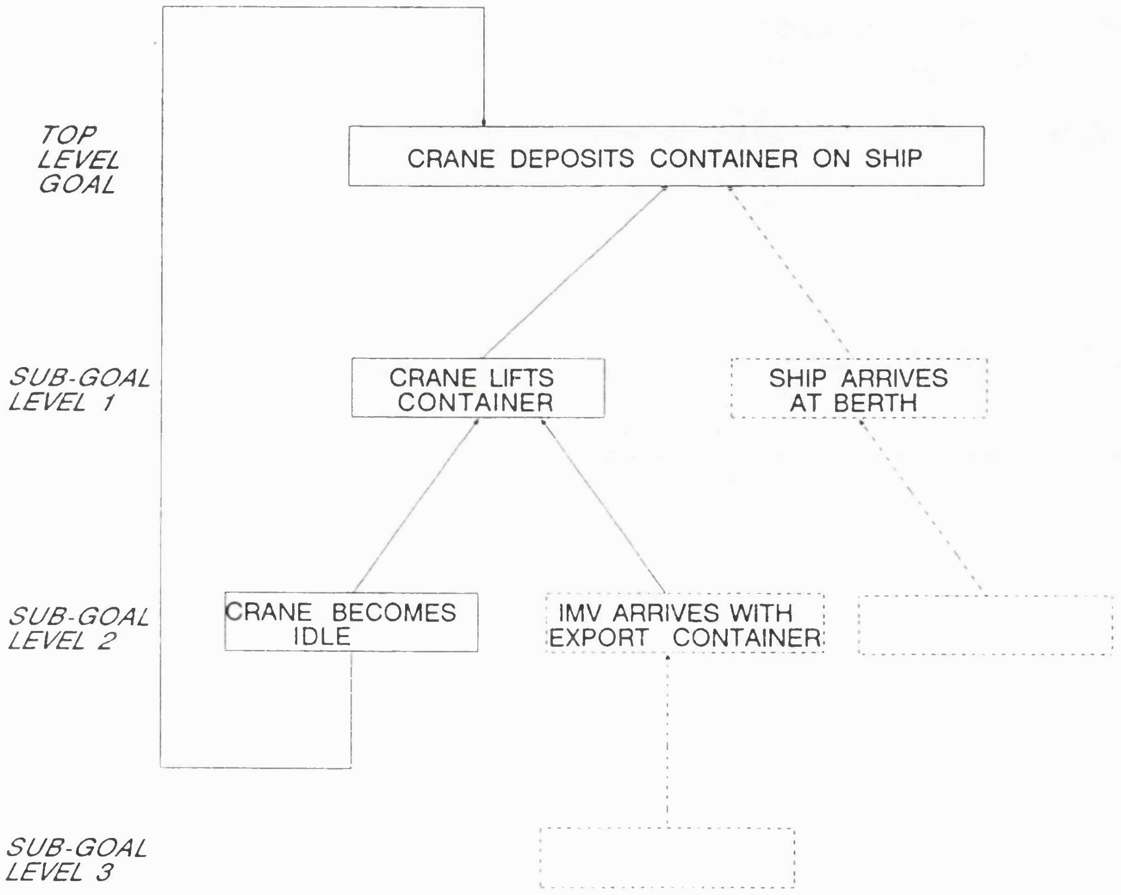


FIGURE 5 SHIP CRANE GOAL

The example in figure 5 shows that in goal driven simulation, the top level goal is never resolved. The inference process consists of a circular reference which drives the simulation until a time or status activated rule finally interrupts model execution.

There are three common approaches to using AI in developing and implementing simulation models: Expert Systems shells which provide a high level, preprogrammed infrastructure consisting of an inference engine and language construct for the definition of the knowledge-base. The second method consists of using AI knowledge engineering tools such as Knowledge Craft (Sathi et al.[1986]), ART (McFall & Klahr[1986]), and KEE (Langen[1985], Jain and Osterfeld[1989]) which provide the necessary code for the representation of knowledge and the implementation of an inference and control process. AI tools provide a greater degree of flexibility as compared to ES shells but at the expense of greater complexity. The third technique makes use of AI languages the most common of which are LISP and PROLOG. AI languages are designed to handle symbolic processing and have built-in features that lend themselves to the development of knowledge-based systems. AI languages are only suitable for use by programmers and in this respect are, in terms of complexity, on a par with conventional high-level languages such as PASCAL and FORTRAN.

1.Using ES shells in developing simulation models: Some attempts have been made at using or adapting expert systems shells in developing simulation models. The limitations of such an approach are numerous and are centered around the generality of the inference processes coupled with very simplistic and non adaptable knowledge representation techniques. ES Shells are designed solely for the implementation of expert systems and do not lend themselves easily to any other type of application.

Moser[1986] describes the use of EXSYS, an expert system shell, in the development of 'business simulation' models. EXSYS was specifically developed with a view to simulation which it is claimed, permitted many of the inherent limitations of the approach to be overcome. Moser's approach does however have a number of shortcomings. The core simulation model consists of a simple FORTRAN program used to solve series of simultaneous equations and is in fact an integral part of the expert system which is itself coded in FORTRAN. The role of the knowledge-base is not one of model representation but rather that of describing rules that establish the value of the model results. Hence, the use of the ES is of no direct consequence to the accuracy and completeness of the model.

Robertson[1986] also adopts a rule-based approach in developing an expert simulation environment. As with Moser, the expert system was specifically developed with a view to simulation. Consequently, the inference engine has a time-keeping facility which, at each time step, scans the rules using a forward chaining strategy. The time advances are achieved by keeping a record of scheduled events and advancing the system clock to the next chronological entry. Model representation is achieved through the use of 'intelligent agents'. Intelligent agents are associated with sets of rules that define their behaviour and an 'agenda' that stipulates the desired goal. Once the goal is achieved, the intelligent agent is 'destroyed'. A goal specified in the agenda can be defined in terms of sub-goals, thus permitting the representation of a sequence of events. Such a sequence is essentially identical to the definition of a cycle of activities in three-phase discrete event simulation. If one further considers the intelligent agents as model entities, the distinction between Roberton's approach and that of Tocher's (Tocher[1962]) three-phase approach becomes blurred, justifying the argument put forward in section 2.4.1 that forward chaining production rule systems are functionally very similar to the executive in three-phase models.

2.LISP based system and Object Oriented Programming: The majority of simulation environments developed in LISP utilise the Object Oriented modelling approach and are implemented on workstations which provide the benefit of powerful graphics facilities for the use of iconics and animated displays. Few environments are coded directly in LISP but rather make use of frame-based knowledge engineering tools which lend themselves particularly well to the object oriented paradigm and provide powerful tools for data input and 'on-screen' model specification. Indeed, some knowledge engineering tools are so appropriate to Object Oriented Programming (OOP) approach that simulation environment have been developed in a fraction of the time that would normally be required. For instance, ART-ROSS (McFall and Klahr[1986]), a clone of the ROSS environment, (McArthur et al.[1986]) was developed using a commercial tool known as ART in under two days and, it is claimed, is an improvement on the original.

Object-Oriented programming is a loose term used to describe a method of knowledge representation based on the description of objects and their interrelationships. The technique originates from the AI field where it is used in developing expert systems, though a similar construct was used in designing the simulation language, SIMULA, in the 1960's (Birtwistle et al.[1979]). Being based on the Expert System(ES) paradigm, object-oriented simulation provides an effective environment for the specification of domain knowledge.

In constructing an object-oriented simulation, the user first creates a set of objects that broadly correspond to real-world objects. The characteristics of these objects are then defined; the inputs they respond to, and the actions they carry out in response. The interplay between objects is represented by the passing of messages. In other words, the action carried out by one object may lead to a message being transmitted to another object specifying that an action should be carried out.

Another important aspect of object-oriented simulation is the concept of 'inheritance' which is derived from the semantic networks knowledge representation scheme used in many expert systems. Inheritance is useful in creating hierarchies of objects, each of which can inherit characteristics from a higher ranking set (Figure 6).

The applicability of the object-oriented paradigm very much depends on the target problem for which a model is to be developed. The OOP approach to modelling is dependent on the entities in the model having a sufficiently close relationship as to be able to establish a hierarchy in which inheritance of characteristics can play a part. The use of inheritance is a key factor in reducing the complexity of the model by limiting the duplication of facts and rules about objects. It is also desirable for the problem domain to be of a type that can be naturally broken down into constituent 'objects' or 'actors' and in which communication plays a significant role. These characteristics are not vital to the model development process, but simplify the overall task by allowing a more natural visualisation of the real world system, that the model is meant to represent. Such considerations are behind the suitability of the object-oriented approach to the simulation of tactical warfare problems to which the ROSS (Klahr[1985]), BLOBS (Middleton and Zanconato[1985]) and SLICE (Gosling and Okseniuk[1986]) languages specifically address themselves.

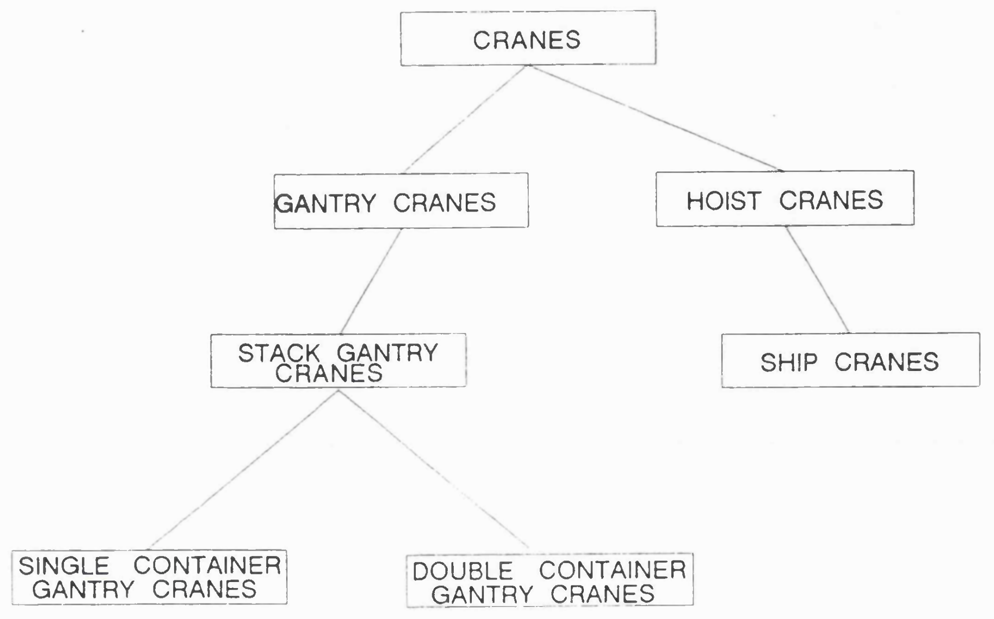


FIGURE 6 INHERITANCE TREE FOR CRANES

In military applications, aircrafts, tanks etc. are effectively described using inheritance. Furthermore, the use of message passing as a form of communication between objects in a tactical warfare problem is a natural means of representing the real-world interplay. An aircraft wishing to land at an airfield can for example be described as sending a message to the control tower requesting permission. The landing activity will then commence, conditional on the airstrip object being available.

Even in the case of applications that would seem suitable targets for an object oriented approach there can be problems such as those identified by McArthur et al.[1986]. Of particular concern is the dependence on message passing for communication and activation of events. Consider the example of two enemy aircraft which are about to go into battle. For one aircraft to recognise and attack the other aircraft, messages need to be transmitted between the two which obviously contradicts the real-world rules of engagement. Another common problem results from the need to represent messages in terms of a limited number of variable values. In most cases, and particularly in military applications, real-world messages are far more complex than can effectively be modelled.

As with expert systems, models developed using the OOP approach are based on a relatively unstructured search algorithm. Furthermore, the desire to allow the user to develop the model incrementally by defining the characteristics of objects as and when they are identified also leads to problems in maintaining a structure. Consequent difficulties also arise because of the problem of ensuring that the defined model is complete and is not ambiguous or inconsistent. Lack of a formal structure also tends to mean that execution is slow for large models which is a problem aggravated by the interpretive nature of the Lisp environment which, as mentioned, tends to be the language used in developing and implementing object-oriented models. This problem was highlighted during the development of I-NET (Reddy et al.[1983]), a corporate distribution and inventory system, using Simulation Craft (Sathi et al.[1986]). The loss in speed is nevertheless partially offset by the advantages of being able to test the effect of changes in the code without having to compile and being able to trace and debug the model interactively.

The use of inheritance can be advantageous in terms of code size by reducing the repetition of characteristics of objects. However, problems can arise when values are inherited unexpectedly. Hence, the characteristics of each member of the object hierarchy has to be carefully defined with particular attention to the possible values that may be inherited from parent classes. Similar care is needed in OOP languages in which rules can be inherited. A set of rules may be spread across a number of object classes making it difficult to trace potential actions and increasing the risk of rules being mistakenly inherited in satisfying a goal. Such problems aggravate the difficulties in specifying the characteristics of the components of the model and particularly in cases where the concept of objects and messages do not seem to be a natural structure for the formalisation process. An investigation of Object Oriented tools and techniques was made as part of the research and is reported in appendix F.

3.PROLOG based systems: PROLOG (Clocksin and Mellish[1984]) is a high-level declarative programming language based on symbolic logic (See section 2.3.1). Facts about objects involved in a problem and rules affecting these objects, are declared and then used in finding a solution without the need to explicitly define a list of instructions.

Prolog's ability at handling rules and representing logical relationships between entities makes it a potential candidate for the implementation of simulation programs. In developing a three-phase discrete event model, rules and facts have to be defined that describe events, entities, and their relationships. Additional rules are then required to handle time advances and storing future scheduled events. Researchers have shown that Prolog can successfully be used in developing simulation models, though the generality of the inference process (and in fact the language as a whole) imposes limitations that remove much of the value of the approach.

Futo, recognising the benefits of using the Prolog approach, has attempted to develop a bespoke version of the language which incorporates within the inference strategy a capacity for combined discrete and continuous time handling. The product of Futo's research, TS-Prolog, uses message passing techniques rather than shared variables for communication between processes. Furthermore, Prolog's backward chaining inference strategy is put to full use by permitting backtracking through time in order to investigate alternative paths through the solution space. Such a goal directed search, which in the case of TS-Prolog, relies on constant activity durations has been criticised by researchers such as O'Keefe and Roach[1987] who argue that the goal of experimentations tends to be unknown or too complex to capture.

Cleary (Cleary et al.[1985]) takes a different approach to Futo in developing T-CP, a modified version of Concurrent Prolog. T-CP attempts to use concurrency in solving multiple goals (asynchronous processes) simultaneously. Each 'process' has its own current state with the T-CP interpreter maintaining a global simulation time. T-CP rules incorporate a 'delay' clause in the form of an arithmetic expression that prevent the rules from reactivating until a point 'delay' units after the last activation of the rule. Cleary has developed, and gives examples of simple models developed using T-CP. These can serve to highlight the drawbacks of using Prolog for simulation modelling.

Prolog does not lend itself particularly well to the development of substantial simulation programs. Clocksin and Mellish[1984], (amongst others) argue that Prolog programs are easily understood by the novice. "...Novice programmers find that Prolog programs seem to be more comprehensive than equivalent programs in conventional languages.". The stated argument is that Prolog code consists uniquely of logical statements describing a problem without the addition of complex and confusing algorithms that specify how the problem is to be solved. Other researchers, notably Muller[1986], argue against this by claiming that declarative programs have to be executed in one way or another giving Prolog a procedural context. "Knowledge of the procedural semantics of Prolog is absolutely necessary for writing correct and efficient programs". Indeed, it is this lack of any explicit definition of what Prolog is going to do with the defined knowledge that makes the creation of substantive simulation model a complex and error prone process.

Prolog, was designed for automatic translation but later used for other natural language applications. Prolog's syntax is based on the notation of predicate logic and uses computational techniques geared towards query handling. Prolog commands are not in an ideal form for the specification of a model, particularly in cases where mathematical operations form a substantial computational overhead. Prolog is being used in an increasing number of AI applications but its design is still geared to closed systems in which inferences are made from defined knowledge with little communication with either the user or the underlying computer system. The consequent support for input/output operations, graphics and external code invocation are limited. Such restrictions reduce Prolog's suitability for simulation modelling given the increasing use of Visual Interactive Simulation, iconic displays, customised device drivers (e.g for mouse control), and mixed language programming.

Another limitation of the Prolog paradigm that applies to the declarative programming approach in general, is speed of execution. Prolog replaces the customised algorithms used in conventional programming languages by a generalised backward chaining inference strategy that seeks to draw inferences and confirm queries using stored information. Prolog therefore spends much time searching for solutions in what may be a substantial unstructured search space. The theoretical simplicity of the declarative programming style is consequently offset by slow execution and lack of control over the inference process. (Shannon et al.[1985])

2.5.5 Interfacing expert systems and simulation models.

Figure 7, identifies the various ways of unifying expert systems and simulation.

Much of the current research in program generators (section 2.5.1), model verification and validation (section 2.5.2), and intelligent front-ends (section 2.5.3) involves some form of cooperative existence between simulation model and expert system.

Another form of expert system/simulation collaboration, in which a conventional model and ES communicate during model execution, necessitates a far greater level of synergy. The remainder of this section is dedicated to a discussion on such systems.

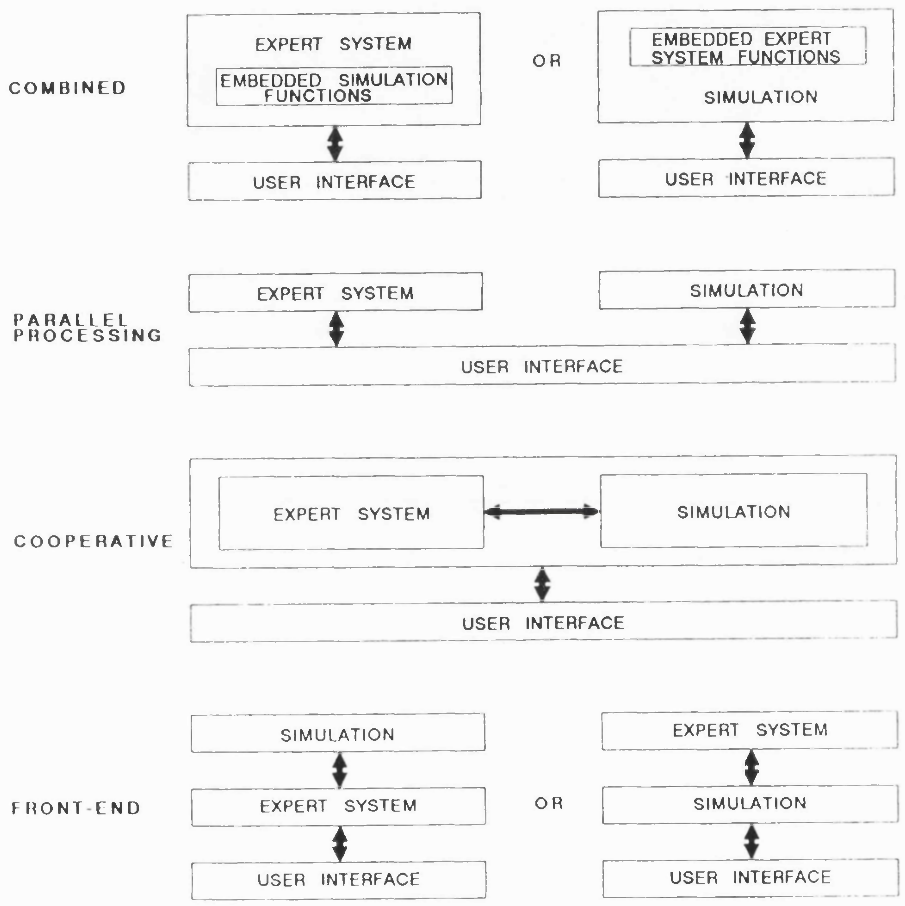


FIGURE 7 COMBINING EXPERT SYSTEMS AND SIMULATION

Researchers including Shaw and Gaines[1986] have highlighted the relationships between simulation and expert systems. "...An ES can be considered as the simulation of the mind of a person whereas most simulation is that of the physical world". One of the advantages in linking a simulation model with an expert system ensues from the ES's ability at providing a model of the experts that communicate and interact with the processes that comprise the physical world. The conventional procedural approach to simulation, based on the use of customised algorithms for the control of the model cannot effectively handle intricate representations of knowledge and decision making. Conversely, expert systems for which knowledge is defined declaratively, and which make use of the generalised deductive capabilities of an inference engine, are not ideal for the representation and definition of physical processes and their interaction. Consequently, simulation and expert systems can be considered as complementary technologies, which, if developed and implemented in parallel, could provide future systems with power "..much greater than the sum of each used separately." (Shaw and Gaines[1986]).

Simulating complex systems such as the port described in chapter five highlights the difficulties in reproducing management decision making tasks that have to be embedded in a model. In some real world situations, expert systems are already in use, assisting management in making operational decisions. Permitting the simulation model to submit queries to the relevant ES, thus bypassing the need to encode the knowledge directly, is of obvious benefit (O'Keefe[1986]). Even in cases where the ES did not formerly exist, benefit would still be derived from developing an appropriate ES in parallel with the development of the simulation model. Once the simulation study completed, the ES could be used separately in assisting in decision making tasks.

Validation of an expert system using simulation forms another class of application for which separate development of expert system and simulation model proves necessary. Stewart and Surgenor[1987] describe the use of a process simulator as a test-bed in the development and validation of an 'on-line' ES for fault diagnosis and crisis management in a production plant. Such an approach to ES validation is conceptually simple requiring an alteration to the ES, changing the source of input from that of sensors to that of simulation output. Stewart and Surgenor implement the simulation and expert system on separate computers, linked via a communications cable. Such a pragmatic approach removes the need for even the most minor modification to the ES since, in any case, input via data cable is the medium used for signal transmission by the sensors. Simulator validation of an ES is more complex in situations where the ES is responsible for managing the safe shutdown of production plant processes in crisis situations by sending appropriate signals to control instruments. In such cases, the simulation model has to be able to respond to input from the expert system, thus necessitating two-way communication.

Flitman and Hurrion[1987] take the same purposeful approach as Stewart and Surgenor and use two computer systems in creating a physical barrier between a Fortran based simulation model and an advisory expert system written in Prolog. The resulting system is used to investigate the possibility of developing an expert system using the method of parameter adjustment. The first stage consists of running the model under user control, allowing the operator to assume the role of a real-world expert. The response actions of the user, triggered by changes in queue lengths, are recorded, formalised as rules, and then stored in the knowledge-base as facts. In subsequent simulation runs, the expert system monitors the simulation output and responds according to the past user interventions.

Reliance on unmanned operations based on the use of robotics and Automatic Guided Vehicles (AGVs) for process automation will inevitably lead to increased interest in expert systems as key components of the Numerical-Control systems (NC). This leads to another potential application for the joint use of expert systems and simulation which, as yet, has not received the level of attention it merits. The ES could benefit from forward projections through time in making a choice between multiple conflicting strategies. Using a simulation model to investigate potential side-effects of a given policy could be of great benefit to expert systems, permitting a higher level of faithfulness with human decision making through the introduction of the added dimension of temporal reasoning.

2.6 CONCLUSION

Simulation is a process that assists a modeller in experimenting with real world situations. Existing simulation structures are well adapted for the purpose of representing the physical structures and associated activities that take place in the real-world. Much of simulation model experimentation is therefore based on assessing the impact of alterations in physical aspects of the system being represented. Existing simulation languages can and are used for the purpose of experimenting with operational decision policies but the difficulties in introducing complex rules and subsequently modifying these rules is a drawback.

The work undertaken by other researchers and reported in this chapter lends credence to the assertion that AI techniques could play an important role in simulation modelling in the context of model experimentation based on changes to complex operational decision policies. It has already been shown by Flitman and Hurrion [1987] that a link between simulation model and an AI language is possible and that such a configuration has a number of practical advantages.

In particular, the expert system "declarative approach" to knowledge definition would seem to have potential in the context of the representation of operational decision making problems in simulation modelling. The apparent advantages relate to the fact that operational policies could be defined in terms of rules without any explicit need to pre-define all program execution paths. A second potential advantage would result from the use of a high-level language which would remove from the modeller the need to delve into complex program code.

The following chapter will focus on examining the nature of decision making and will attempt to analyze what is entailed in the representation of operational staff and production management. Aspects of decision making such as "joint" problem solving will be examined and it's impact on the requirements of a simulation modelling environment considered.

CHAPTER THREE

REQUIREMENTS OF A DECISION ORIENTED SIMULATION ENVIRONMENT

3.1 INTRODUCTION

The design of the proposed simulation environment is dependent on how decision related knowledge can best be integrated with a corresponding simulation model depicting physical activities. A first step in establishing the most appropriate design is to investigate the difficulties in representing decision tasks. Section one therefore consists of an analysis of decision making with emphasis placed on the classification of decision types according to position in an employee/management hierarchy. Although such a hierarchy is not typical of all problem areas investigated using simulation, the environment is one of the most complex types to represent. The section concludes with a study of the data analysis tools used by decision-makers. For accuracy, some of these tools may have to be represented as part of a model.

Simulation models presently depict decision processes by either simplifying decision rules or by replacing the logical steps involved by probability distributions based on prior sampling. In cases where the actions taken by decision-makers are highly predictable, and where experimentation using alternative decision rules is not deemed necessary, existing simulation environments operate satisfactorily. In other cases, such approximations are considered to affect the accuracy of the simulation and limit the potential of the model as a tool for experimenting with alternative decision criteria. Present simulation environments do not have the capacity to represent aspects of decision making to the level of detail described in section one. Section two consequently considers the limitations of simulation in representing decision-makers and investigates Visual Interactive Simulation (VIS) as a means of overcoming such problems.

An expert system is used as a supportive tool in reproducing an expert's solutions to a range of problems. In contrast to simulation modelling, little emphasis is given to the accuracy of the internal representation. Many expert systems use production rules which do not necessarily reflect the kind of deductive reasoning that the expert applies. Secondly, such rules are often randomly ordered making the knowledge-base harder to maintain and reducing the efficiency of the search process. Such problems, coupled with an inability to define the knowledge of several decision makers in the form of an employee hierarchy, limit the value of expert systems both generally and in the context of simulation applications. Section three considers the impact of such limitations in more detail. Isolating the potential problems in using existing expert system methodologies is necessary in determining the most appropriate approach to developing the proposed simulation environment.

Whereas greater detail is necessary in the representation of decision-makers, a careful balance must still be maintained between accuracy and development overheads. For instance, inclusion of representations of high level management may be unnecessary given that decisions taken at this level tend to apply to overall company policy or strategies and tend to only take effect in the long-run. Section four is devoted to an analysis of such considerations in isolating the requirements of the proposed simulation environment. Potential system designs are then discussed relating to the simulation language, the expert system, and their possible amalgamation.

3.2 DECISION MAKING.

Decision making generally involved making a choice from a range of alternatives based on a specific selection criteria (Nestman and Windsor[1985]). The selection process typically involves the comparison and analysis of operational data and may necessitate substantive skill and experience.

Decision making activities take place at all levels within an organisation, from the chairman of a company down to shop floor employees. In general, the decision making activities and goals pursued by senior management are hard to identify and can be defined as being unstructured whereas decisions and goals made by low ranking employees tend to involve operational activities that are easier to describe, are well structured, and most often well documented. Furthermore, the range and diversity of data used by senior management tends to be far greater than at the production line level where, for instance, an operator bases his decisions on machine data and instructions obtained from the line manager. The passing of data to a more senior colleague and the response obtained is a critical aspect of decision-making that occurs at all levels in an organisation. The communicative aspect of decision-making is represented diagrammatically in figure 8.

Decision domains have been classified by Nestman and Windsor[1985] as belonging to one of four categories: Correlative, strategic, tactical, or operational. Correlative and strategic decision domains are investigated solely by high ranking management in the processes of planning and goal setting. Analytical tools used are of a 'qualitative' nature and are either heuristic or deductive in nature. Heuristic tools include fuzzy set theory, intelligent delphi and catastrophe theory. Deductive tools consist amongst others of, simulation, contingency planning, and markov processes. Qualitative tools are imprecise in nature but are the only effective techniques available to the manager. In contrast, tactical and operational decisions tend to be taken by middle management and operations staff using 'quantitative' tools. Quantitative tools are either statistical or algorithmic in nature and are sufficiently precise to be of considerable use in decision making activities.

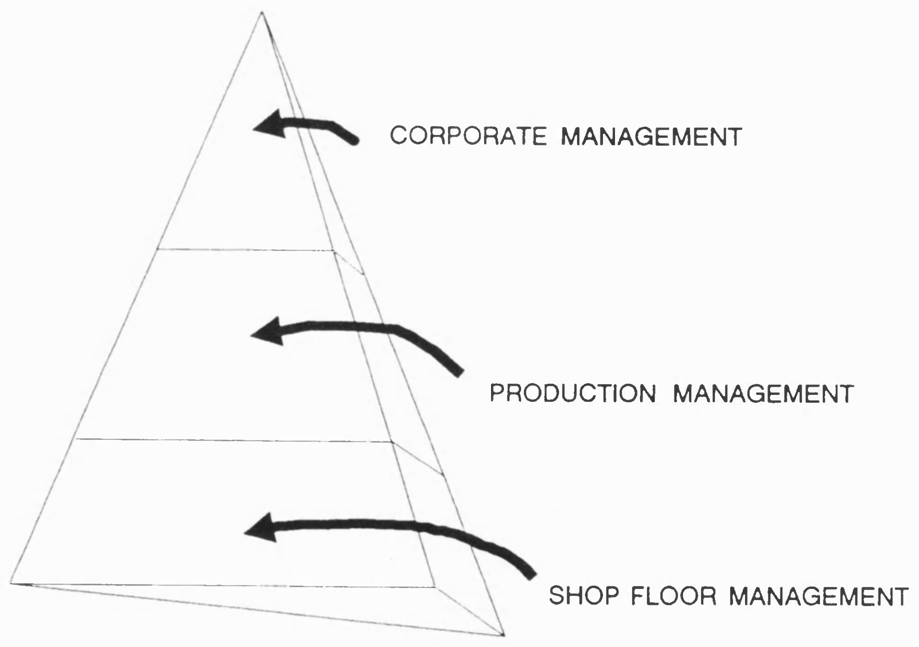


FIGURE 8 RULE-BASE MANAGEMENT HIERARCHY

Decisions can further be categorised as either being of a pre-emptive or corrective type. Pre-emptive decisions are applied in ensuring that undesirable situations do not occur. They act as physical constraints. e.g. machines overheating, dangerous power consumption levels etc.. In contrast, corrective decisions are only applied after an undesirable event has occurred. Both Pre-emptive and corrective decisions tend to be applied by operations staff and consequently tend to make use of the quantitative tools described.

3.3 DECISION MAKING WITHIN SIMULATION

In section 2.2.5 and 2.5, the impact of simulation on decision making in terms of decision support systems was considered. A different viewpoint will now be taken by investigating what decision making activities need be represented as part of the model and how these can effectively be incorporated into the logic framework.

According to Rozenblit and Zeigler[1985], 'conventional simulation languages are limited by the necessity to settle on fixed, simplistic resolutions to a number of complex tradeoff decisions'. This observations is also reflected in a discussion of the comparative merits of simulation and gaming in decision support systems by Gray and Borovitz[1986] where simulation is characterised as appropriate in cases where the systems involved only limited or highly predictable human behaviour.

A common approach in simulation is to model behaviour using probabilities based on prior observation and sampling. Alternatively, simplification is achieved by using crude and simple decision rules that scarce do justice to the complex and adaptive behaviour being applied in the real world system. In contrast, physical components of such systems are typically represented to high levels of accuracy. O'Keefe and Roach[1987] believe that this inability to effectively model intelligent behaviour is one of the major drawbacks of traditional simulation modelling. The argument is that if the representation of potential decision processes is crude and inadequate, then any comparison of decision alternatives will inevitably be restricted to a range much narrower than that available to the decision-maker in practice. Such a drawback is particularly restrictive in tactical simulations (e.g warfare) in which the main purpose of the simulation is to make such comparisons.

Researchers who have considered the need to improve the methods used in defining simulation models have tended to investigate the possibility of developing alternative model building techniques. The primary concern has been to simplify the model building process rather than add to its complexity by increasing the detail with which decision-tasks are represented. One approach that has attempted to compromise model detail with modelling complexity and that seems to have had some degree of success is Visual Interactive Simulation (VIS). VIS represents changes in the state of the model in pictorial form during the simulation runs (see O'Keefe and Roach[1987]). The user is able to interrupt the simulation, alter a restricted range of model characteristics, and then continue the run using the modified model. By appropriate interventions in response to observed model behaviour, decision mechanisms of arbitrary complexity may be achieved. However, this achievement is at the cost of not being able to secure the benefits of replication and statistical analysis of performance. VIS modelling is essentially a hybrid of simulation and gaming, sharing a measure of the advantages and disadvantages of each.

The approach adopted in this thesis is a more direct and obvious response than VIS to the challenge that simulation models do not adequately represent intelligent behaviour. The area of artificial intelligence has been a natural focal point in the search for an appropriate representation of intelligent behaviour for incorporation in models. This investigation of the field of AI, and in particular expert systems, has led to a number of possible designs for the implementation of the proposed simulation environment. These will be discussed in some detail in section 3.5.

3.4 REPRESENTING DECISION MAKING USING EXPERT SYSTEMS.

Clearly, the principle demand on the nature and structure of an expert system in representing human decision-makers is that it should be capable of an adequate representation of the individual decision-making processes exploiting such information as would normally be available to the individual. This information is both current and historical and for instance, may be based on an investigation into the future using a stochastic tool such as simulation as was described in section 3.2. Consequently, if a decision-maker has access to a simulation model, then an expert system that is to adequately represent the decision-maker must have access to the same source of information. An expert system may have to be able to represent time for other reasons. For example, decisions taken in a dynamic system must take into account delays both in the information flows on which decisions are based and in the actions taken under those decisions. The use of temporal reasoning in expert systems has been discussed by Miller[1986] and researched in a more practical context by Fox and Smith[1984] during the development of ISIS. ISIS is an expert system for scheduling in job-shops and makes use of heuristics in generating alternative schedules. ISIS, it is claimed, has a performance level superior to that of its human counterpart.

A number of other characteristics of decision making limit the effectiveness of existing expert systems. For instance, the satisfactory representation of individual decision-makers cannot be easily separated from the need to represent their interaction. One cannot sensibly represent the decision making activities of an employee without reference to the contributions made by others which may influence the outcome of the decision. Even senior management decision making may indirectly influence the outcome of decisions taken by low-ranking employees through the modification of global goals or through alteration of the methodological aspects of the decision making process. Balmer et al.[1988] recognise this problem and suggest a methodology based on the definition of 'Epi-rules' that reflect the influence that management rules have on other decision making processes (see section 3.5). Current expert systems ignore the impact of senior management and take a simplified view of even the most basic decision making processes. A modified expert system paradigm that takes into account the existence of a hierarchical management structure would in some sense complicate the representation of management expertise, though the resultant expert system should benefit from it.

Expert systems take a simplistic view of decision making which can partially be attributed to a number of factors associated with the design of ES development tools. Most expert systems are poor at handling arithmetic and algorithmic computations which complicates the representation of the 'quantitative' tools used in making tactical and operational decisions. As mentioned in section 3.2, middle management and operations staff make use of such statistical and algorithmic tools in decision-making, and any difficulties in representing these clouds the effectiveness of the expert system paradigm.

Expert systems also lack the necessary structural formalism that would permit the representation of individual decision-makers and their interaction. The development of expert systems for medical diagnosis such as Mycin and Internist owe much of their success to the restricted, 'one man' view they take of the real world. An expert system to aid decision making in a manufacturing environment may necessitate a far greater level of complexity given that employees ranging from machine operators to line managers contribute to the decision making process. Existing expert systems encourage an incremental process of knowledge-base development with elicited rules added in a random fashion. There is a lack of any formal methodology facilitating the physical separation of logically distinct rules attributable to individual decision-makers. The maintenance and validation of large rule-bases is consequently made more difficult, with the addition or deletion of rules potentially leading to unexpected results.

3.5 LINKING SIMULATION AND EXPERT SYSTEMS - A SUGGESTED APPROACH

In section 3.3, the drawbacks of traditional simulation were highlighted in terms of the inevitably simplistic nature of modelling resulting from difficulties associated with representing the process of decision-making. In section 3.4, expert systems were shown to lack the structural formalism and algorithm handling capabilities necessary in representing decision-makers and their interactions. In this section, suggestions are made as to ways of overcoming these problems. The proposed methodology is based on the integration of expert systems and simulation to form the backbone of a system devoted to decision support. The suggested approach is intended to help integrate management decision rules in the simulation by facilitating the representation of the dynamic aspect of decision-making, and the modelling of the interplay between individuals during decision taking activities. The resulting methodology should provide a composite technical base for a decision support environment in which experiments can be carried out based on the analysis of the effects of differing responses to decision making tasks.

Adequately representing decision-makers and employee hierarchies is an intractable problem for those currently working in the area of simulation (sections 2.2.4 and 3.4). Increased faithfulness to the real world is required permitting the individual specification of employee decision rules and the sharing of such defined knowledge. The management hierarchy in a manufacturing environment will include many persons whose role in decision-making is crucial but who are not normally represented in a simulation model which will usually concern itself only with those actively engaged with machine operations. The modelling of decision-making must extend to this hierarchy and will sensibly reflect its structure. The immediate effects of higher level management decisions may be seen in terms of modifying either the goals or the decision making processes used by those at lower levels rather than acting directly on the system.

The representation of corporate management decision making, for the most part, is not crucial to the simulation. Decisions taken at this level mostly apply to the long-run and do not tend to have a direct influence on the day-to-day operation of the system. Production line management are most likely to be the hardest to model as communication with lower ranking employees is likely to play a major role. Decisions, though not instantaneous, are typically carried out within the duration of a simulation run and consequently effect the outcome.

Decision making in the real-world is a dynamic process and not instantaneously as represented in expert systems. The delays are particularly critical when methods of communication are limited. The representation of time delays in the expert system may therefore be of importance. The ES should be able to represent time in two way:

> Decisions that are made now but only take effect at some time in the future. Hence, the decision-maker is using information currently available to him. But as the actions only take place at some point in the future, the data available will have changed when these take place.

> Decisions that are made now, to make decisions at some point in the future. Hence, when the action(s) finally take place, the decisions are based on currently valid data. For instance, a manager may say to an employee that he should apply certain rules at a given point in the future.

In section 3.4, it was noted that from the expert system perspective, that the representation of decision-makers relied in part on an adequate description of the decision making processes coupled with access to such information as would normally be available to the individual. As we have seen, the process of decision making may involve the use of statistical and algorithmic tools which are not easily defined using the declarative style of knowledge representation. Ideally, the expert system should have access to code written in a conventional high-level language in achieving a greater faithfulness to real world decision taking tasks and giving the expert system a combined declarative/procedural context. The other aspect of accurately representing decision-makers, namely, providing the expert system with information normally accessible to the decision-maker is achievable by treating the simulation model of the environment in which the individual operates as a form of data generator. In turn, the expert system may effect an appropriate series of actions by returning any decision reached to the simulation model. The information available to the expert system could derive from the current and historic data of the simulation model and be made accessible by some sharing of data structures. However, access to data by the decision-makers represented in the expert system must be restricted according to the range of data accessible in the real world. Modelling must include the definition of the data which should be made accessible to the expert system components and that which must remain privy to the simulation. For instance, the simulation structures will typically include some future event list. Clearly whilst access to information concerning model futures could be most useful, decision rules depending on such 'clairvoyance' could not be regarded as legitimate and should be formally excluded.

Increasing the level of detail in a simulation model may complicate its use as a basis for experimentation and limit its applicability in the context of decision support. Consequently, the developed environment must provide the necessary tools and be structured in such a way as to simplify the development of a model. The process of validating and using a model, and interpreting results obtained must also be aided through the use of appropriate tools. The expert system component of the model may be used to improve the user-friendliness of the environment by controlling the user interface or by encapsulating the necessary logic for the analysis of the simulation results. The expert system should also improve the maintainability of the model by simplifying alterations to the model logic. For instance, the environment should support modification of rules applied by decision-makers without a corresponding need to adapt model activity cycle representations. Model maintainability is further discussed in the next section.

3.5.1 Integrating expert system and simulation methodologies.

Several techniques have been adopted by researchers in attempting to use simulation and AI to mutual benefit during the modelling process. These approaches were described in chapter two. The merits and disadvantages of each will now be re-considered with the aim of identifying the most appropriate design for the intended decision support environment. The desired characteristics of the simulation language outlined in the last section are taken as major factors influencing the design of the eventual system.

1. Using expert system shells: In section 2.5.4, the use of an expert system shell as a form of simulation model development environment was considered.

The main argument put forward in suggesting that an expert system shell may be suitable for simulation is based on the similarities between simulation and expert systems identified by Shaw and Gaines[1986] and Doukidis[1987], amongst others. The likeness between the three-phase simulation 'executive' and the expert system forward-chaining inference engine are considerable. Indeed, Robertson[1986] and Moser[1986] have shown that an expert system can be adapted for simulation. The fact that the expert system and simulation paradigms are similar is not however sufficient to argue that an ES shell provides an improved environment for simulation. Using a shell would only be of benefit if the inference engine could be made to handle both the simulation and the conventional expert system task of representing decision-making. The inference engine is not capable of this dual role and the language syntax used in shells is not sufficiently flexible to permit the separate specification of knowledge relating to decision-making and knowledge specific to the definition of activities and their relationships. Other limitations of the approach are listed in figure 9.

ADVANTAGES	DISADVANTAGES
Simple, declarative structure.	Limited/non-existant algorithm handling capability.
Intelligible language syntax.	Limited arithmetic and data handling facilities.
Traceable inference.	Unstructured language syntax.
Facilitates incremental development.	Slow code execution.

FIGURE 9 ADVANTAGES/DISADVANTAGES OF EXPERT SYSTEM SHELLS

2. Creating a new simulation language: Concepts borrowed from expert systems and simulation can be brought together using a common language syntax, representing another possible methodology for interfacing a model with a representation of decision-makers and their hierarchy. The main advantage with this approach is in the flexibility afforded by the creation of a bespoke language. Commands can be structured to match a given purpose, data structures are compatible, and facilities can be provided for manipulating stored information in a way consistent with the task in hand. Examples include modelling environments based on Object Oriented Programming (OOP) and modified versions of Prolog (section 2.5.4). In OOP, a different view of simulation is adopted in which knowledge of the real world is defined in terms of objects and their interrelationships. No distinction is drawn between knowledge concerning the physical objects that make up the real world environment and knowledge specific to the process of decision making. In Prolog based environments, the forward chaining inference process can be modified so as to represent discrete advances in time. The basic Prolog syntax however remains unchanged with the advantages of a common language syntax consequently being lost.

One of the drawback with using a single bespoke language in specifying the expert system and simulation model results from difficulties in merging knowledge concerning model activities with knowledge specific to decision-makers. The physical actions or processes that constitute a major part of existing models are best defined using sequentially executed code. On the other hand, decision related knowledge is ideally specified using a declarative language. The two language types cannot effectively be merged without compromising on the effectiveness of either the simulation or expert system.

The creation of a single bespoke language is also undesirable from the perspective of code maintainability. For instance, the simulation model of the port described in chapter five is of a size that limits the practicality of making changes to the model logic. The individual cycles of the model are defined in separate modules to ease the interpretation and handling of the code. Had it been possible to add complex management decision rules to the same program, the maintainability and legibility of the code would have been lost. Indeed, the difference in nature between the two types of knowledge would have provided the incentive to place the decision rules in a separately identifiable module. The benefits and limitations of creating a bespoke simulation language are summarised in figure 10.

<u>ADVANTAGES</u>	<u>DISADVANTAGES</u>
Code may execute faster.	ES not separable.
Code modification limited to one file.	Logical separation is lost.
May be easier to debug.	Risk of inconsistencies.
Bespoke language syntax.	

Figure 10 Advantages/disadvantages of a bespoke simulation language

3.**Interfacing an expert system and simulation model:** In section 2.5.5, an analysis was made of systems developed by Stewart & Surgenor[1987] and Flitman & Hurrion[1987] in which it was seen that limited interaction was possible between a separately implemented simulation model and expert system. Both environments were designed using the pragmatic approach of separating the simulation model and expert system using two microcomputers. Data transfer was achieved using a general communications protocol. The same principle could be applied in separately specifying knowledge pertaining to model activities and decision-making. Not only would the code be easier to interpret but the logical separation would enable the physical model to be modified independently from any changes made to the decision making activities. The barrier between the software modules would encourage parallel development and testing using tools that differ in characteristics and would help in retaining some logical structure to the implemented model. In many respects, separation of the knowledge from the physical implementation follows the expert system paradigm and corresponds to the concept of keeping the inference engine separate from the rule-base.

Stewart and Surgenor's approach is limiting in the sense of having to use two microcomputers. Apart from the physical restrictions imposed, the need to transmit relevant data over a communications cable removes much of the potential value of interfacing a simulation model and expert system. For instance, the expert system module must transmit a request to the simulation model in order to examine an item of data. The data returned is limited in terms of volume and must be preceded by a message identifying the request. An alternative method of communication is based on the simulation model transmitting of its own accord data deemed to be of interest to the expert system. The 'broadcast' approach is limiting in terms of the volume of data that need to be transmitted. Further requests for data may also be necessary if information is found to be lacking. Extreme care is also necessary in ensuring that data held by the simulation model is identical to that being used by the expert system. Any inconsistencies that may occur would invalidate model output. The problems would obviously be compounded if two-way data transmission were to be introduced.

The benefit of using communicating microcomputers is in the flexibility afforded by being able to use existing software with relatively minor adaptations. In practice, incompatibilities in data types and variable storage structures impose considerable restrictions on the practicality of the approach. Alterations to data formats are necessary for every item of information sent from one computer to the other. More critically, computations based on values stored in data structures such as arrays or series of records are rendered impracticable by the necessity to transfer all relevant data items. The process is further complicated if the data structures used on each computer differ. For instance, Prolog's use of Lists which can contain mixed data types does not have a direct equivalent in procedural languages such as Pascal, Fortran, and C.

A more satisfactory approach than using two microcomputers is to create an environment entirely based on a single machine. Two potential methodologies have been considered and will now be discussed in turn.

A simulation language could be interfaced to an expert system by keeping both modules memory resident and using pre-defined routines for transfer of control. The simulation would then call the expert system when particular decisions needed to be made. The expert system could either be based on a shell such as Xi, an AI tool (e.g. OPS5), or an AI language such as Lisp or Prolog. The principle benefit of the approach would be that existing software could be used that has already been extensively developed and tested, thus limiting implementation overheads. Using a single computer has clear functional advantages over Stewart and Surgenor's approach. Nevertheless, incompatibilities in data types and storage structures could not be overcome, and some form of communications strategy would still have to be established. At the outset of the research, the approach was considered a possibility as it was found that the language specification of Borland's Turbo-Prolog appeared to allow linking to Pascal and C. Turbo-Prolog produces object code which should be linkable to other object code modules produced using other language compilers. Thus mixed model programming should have been achievable bringing together elements of AI and simulation code written in a conventional procedural languages. Borland's claims to produce Microsoft compatible object code could not however be born out in practice and the idea had to be abandoned.

The second potential methodology, is to write an expert system shell in a high-level procedural language. The creation of a bespoke expert system would mean that the software could directly be developed for use with a simulation model. The main drawback of the approach would be the time overhead necessary in developing the expert system. Nevertheless, the benefits to be derived are clearly substantial with problems such as the incompatibility of data structures potentially being solvable. Compatibility in data types and variable storage structures could be achieved by using a common language in the development of the expert system and simulation environment. Extensive research has already been carried out at the LSE in the creation of Pascal routines for discrete-event simulation. Hence, if these routines were to be used, the expert system shell would also have to be written in Pascal.

The development of a simulation environment and separate expert system written in the same language would help relieve the problem of data compatibility. Nevertheless, the greatest obstacle to the creation of an effective system would remain. Namely, the need to multitask the programs and physically transfer data between the systems as and when required during the execution of the programs. The only apparent way of overcoming such limitations would be to create a single executable program combining aspects of the simulation environment and expert system. The simulation and ES could then share variables and other relevant data via the stack, heap, or any commonly agreed memory locations. The independence of the units would nevertheless need to be retained to permit separate development of the model and knowledge-base. This requirement linked with the fact that the eventual system is potentially considerable in size, points to the need to adopt a modular approach. A programming language that permits the creation of independent object code modules would satisfy these requirements, allowing a single executable program to be created by linking the individual modules together.

Figure 7 and 11 summarise the alternative ways in which an expert system and simulation model can be interfaced.

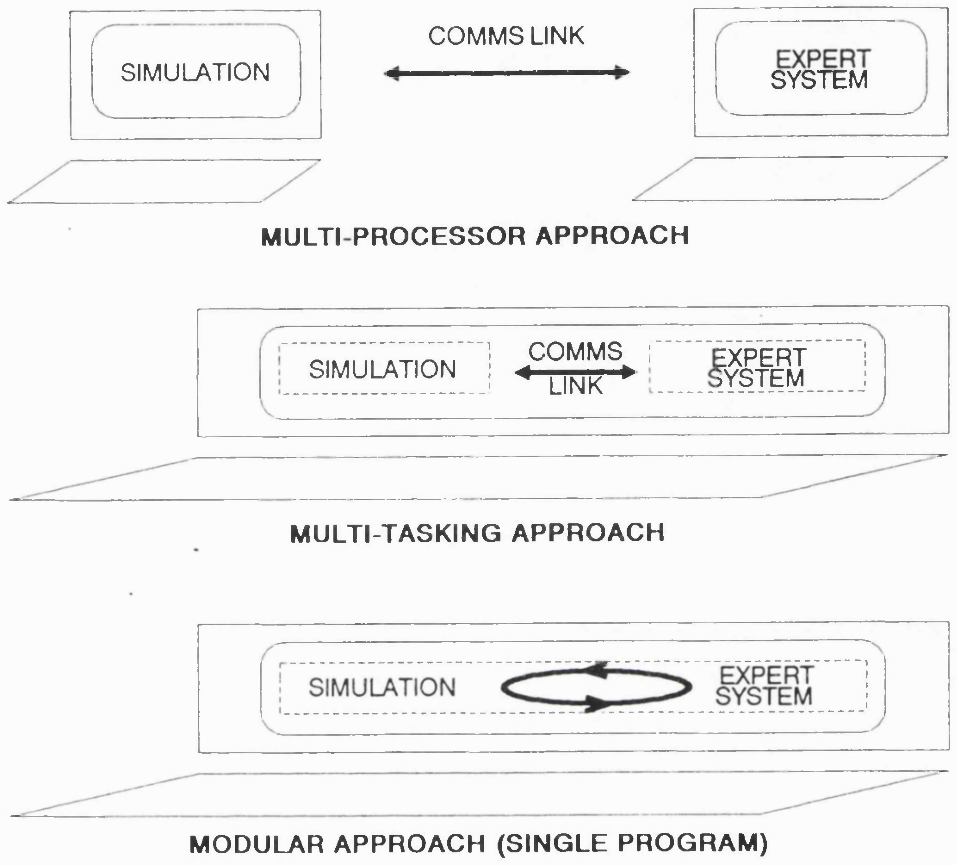


FIGURE 11 ALTERNATIVE APPROACHES TO SIMULATION / ES INTEGRATION

3.5.2 Facilities that should be provided by the expert system.

In section 2.3.3, the limitations of expert systems were considered. These are necessarily relevant to the problem of interfacing a simulation model and expert system, and are currently major issues in artificial intelligence research. The scope of the thesis cannot include the improvement of the expert system paradigm. Nevertheless, some progress may be achieved indirectly through the integration of simulation and AI techniques, and the need to represent multiple decision-makers within the knowledge-base.

The use of an off-the-shelf expert system shell would be limiting in a number of respects:

1. Many of the facilities provided would be redundant.
2. The shell could not be customised.
3. Interfacing the expert system to other code would be a complex process and would inevitably effect the usability of the end-product.
4. Commercial shells produce screen output. This would interfere with the output produced by the simulation model. (Another reason justifying the use two computers)
5. Expert system shells provide poor support for arithmetic computations and most often do not support execution of procedural code.

The proposed development of a bespoke expert system would help in overcoming these problems.

Only those facilities considered necessary would be implemented as part of the expert system. The compactness of the resulting code would help improve the efficiency of the inference strategy and reduce the problem of limited Random Access Memory (RAM). Code size is particularly critical if the system is to implemented on an microcomputer.

A bespoke expert system can of course be customised. The format used for the specification of rules can be adapted to the specific needs of the simulation environment. Most critically, the ES could be made to share variables and data structures with the simulation model.

The use of Pascal as the base-language for the expert system is helpful in respect of taking full advantage of the facilities provided by the underlying computer hardware. Transfer of control between the ES and simulation model should be greatly simplified as should be the necessary task of accessing memory locations.

The bespoke expert system permits full control of the screen display. Hence, output produced by the simulation model could be windowed with output generated by the expert system. Alternatively, each system could be prevented from sending information to the monitor simultaneous to the other.

The arithmetic support could be customised as necessary. Furthermore, the expert system should be capable of calling Pascal procedures and functions. This would give the ES the added benefit of being able to execute procedural code. Any complex algorithms used in decision-making problems could be defined using functions.

The development of a bespoke expert system would also permit the customisation of the language syntax. Using the ES in a simulation context imposes requirements on the system that differ from those that would be encountered in developing a conventional ES. For instance, the knowledge-base has to be capable of representing several decision-makers whilst keeping some form of logical separation between the sets of rules. Ways of overcoming the problem include using individual files to represent each decision-maker, or creating blocks of rules within one file. In section 3.4, the issue was taken a step further by pointing out that decisions are not always taken by lone individuals but often by two or more people in consultation with each other. A potential way of limiting such representational problems would be to permit each block of rules to 'inherit' rules from other blocks. Hence, if the rules used by one expert were not sufficient to solve a given problem, another expert's block of rules could also be used.

Finally, the adequacy of the expert system relies in great-part on the design of the inference strategy. Expert systems either use a methods based on forward-chaining, backward-chaining, or a combination of the two. These were described in section 2.3.1. Forward-chaining inference strategies are data-driven. In other words, the application of values to the premises of rules is what triggers their execution. Consequently, if forward-chaining was to be used in the proposed system, the ES would have to assume the role of a 'data-analyser'. Whenever a change in state occurred in the simulation, the ES module would be activated and all rules verified to see if their premise could be satisfied. In a sense, the use of forward-chaining would help in creating a realistic representation of methods used in the real-world. For instance, managers at the control of a manufacturing plant are continuously monitoring the production line, taking appropriate action when undesirable events occur. The main disadvantage with the approach is the computational overheads incurred in having to scan all rules stored in the knowledge-base at every time advance. The process is time consuming and inefficient as the majority of rules will not apply. The alternative backward chaining strategy is goal-directed. The expert system is presented with a goal which it then attempts to prove or disprove. The simulation model has to initiate the inference process and specify the goal to be resolved. For instance, during the C-phase of the three-phase approach, a search is made for activities that can commence. The conditions that dictate whether the activities can start could include references to the expert system. In the manufacturing plant example, the closest analogy would be a machine operator checking whether he could start a particular process. The strategy is more efficient in that the search is directed with irrelevant rules being ignored. The division of rules into blocks should also help by further reducing the search space.

3.5.3 Facilities that should be provided by the simulation component.

As with the expert system, modifications need to be made to the simulation language in providing a capacity to communicate with other software. Access to source code is therefore indispensable. Research at Lancaster university and subsequently at the LSE lead to the creation of the Extended Lancaster Simulation Environment (eLSE), a set of library routines for the development of discrete event three-phase simulation models (Chew[1986]). The routines are written in Pascal, are well tested, and fully documented. The eLSE libraries could be modified, thereby by-passing the need to create a new environment.

One of the most basic requirements of the simulation module will be to support models that are substantial in size. The addition of an expert system will place substantial burdens on the simulation environment. The eLSE libraries are currently implemented in Turbo-Pascal which imposes a limit of 64K on code size and is consequently inadequate. The libraries therefore have to be transferred to another version of Pascal. Other necessary changes include the addition of routines to transfer control to the expert system module. These library routines have to be capable of passing information over to the ES and receiving instructions in return. In the case of a backward-chaining system, such information would include a definition of the goal to be solved. The instructions which are subsequently received from the ES, such as an indication as to whether an activity should commence, have to be interpreted and carried out.

3.6 CONCLUSION.

The problems that would be encountered in creating a formal representation of decision related knowledge were highlighted by considering the nature of decision making. The employee hierarchy in an organisation was taken as an example. The knowledge applied and goals pursued by senior management were shown to be difficult to isolate given the unstructured nature of the decision activities. In contrast, decisions made by machine operators were identified as being well structured and consequently far easier to formalise as program code. Decision types classified by Nestman and Windsor[1985] were then discussed. The tools used in decision-making were also examined and categorised according to their precision. In conclusion, it could be seen that part of the difficulty in representing the decision-making activities of management was the underlying need to reproduce the characteristics of the heuristic or deductive tools being used. In contrast, formalising the tactical and operational decisions taken by operations staff would be facilitated by the well defined statistical or algorithmic tools being applied.

The limitations of simulation techniques in formalising decision-making processes were then considered. Simulation was characterised as supporting a detailed representation of the physical components of real-world systems but could not effectively be used to model intelligent behaviour. Techniques that are applied in representing decision tasks include simple rules that only partially reflect the underlying logic and sampling from probability distributions. Such approximations, if reasonably accurate, need not invalidate results concluded from simulation experiments. However, if the decision-making process is complex and difficult to approximate, the output obtained may be inaccurate and bring into doubt the value of the model. Simplifying the model logic also effects the flexibility of the simulation. The potential range of experiments that could be carried out in comparing decision alternatives is inevitably restricted to a range much narrower than would be possible if the model depicted decision-making in detail. Visual Interactive Simulation (VIS) is a technique that helps overcome the drawbacks of limited model detail in the representation of decision alternatives. VIS allows the user to interrupt and modify a model during a simulation run, thus reflecting the decision-makers action. Such a benefit is however secured at the expense of not being able to replicate experiments or carry out statistical analysis of performance.

Section 3.4 then considered the appropriateness of the expert system paradigm in representing decision-makers. Expert systems were identified as being insufficiently powerful to represent the intricacies of the decision-related knowledge of high-level management. This was seen as being largely due to the difficulties in representing the heuristic and deductive tools used by management as part of the knowledge-base. Expert systems were also seen as imposing a limit on the representation of multiple decision-makers. In the real-world, decisions are frequently taken by groups of people rather than by individual in isolation. Sharing knowledge consequently forms an important part of the process of taking an action. Expert system shells tend to be developed with a view to representing a single expert and do not have the necessary structure to represent several experts each contributing in some way to a given decision task. Finally, expert systems were seen to impose some limitations in situations where mathematical computations were involved. In the context of decision making, operations staff may use statistical or algorithmic tools as aids in deciding on an appropriate course of action. These tools may be difficult to represent as part of the knowledge-base.

Alternative designs were then considered in bringing together aspects of simulation modelling and expert systems. The main concern was to identify the most appropriate approach to adding decision related knowledge to the model. The first step consisted of reviewing the implications of having to represent the employee hierarchy and the consequent sharing of information. Formalising the techniques applied by senior management in solving decision related problems had been identified in section 3.2 as a major problem. After careful consideration, it was concluded that this inability to effectively represent corporate management would not adversely effect the validity of the simulation model. Decisions taken by corporate management tend to consist of long-term strategy related problems which would not typically take place within the time span of a simulation run and could not consequently effect the outcome. The techniques used in developing the state-of-the-art modelling environments described in section 2.5 were then discussed. It was concluded that the approach that presented most promise, consisted in being able to specify decision rules in a knowledge-base, separate from the rest of the model. Having focused on this approach, alternative designs were considered for interfacing the simulation and expert system. Stewart and Surgenor's[1987] research in validating expert systems was then examined in evaluating the potential of their approach of using two interacting microcomputers for implementing each of the software modules. The need for a complex communications protocol and the consequent difficulties in sharing common data were found to be critical limiting factors. The ensuing conclusion was that the simulation and expert system components had to be implemented on a single computer and that the modules had to effectively be combined. The last section of the chapter highlighted the desirable features of the expert system and simulation model components, finalising the broad definition of requirements for the proposed environment.

CHAPTER FOUR

ESSIM - AN ENVIRONMENT FOR SIMULATION

4.1 INTRODUCTION

Previous chapters have served to define the requirements of an advanced modelling environment encompassing aspects of simulation and AI. This chapter will describe the development of a number of prototype designs that culminated in the development of the final system named ESSIM (Expert System SIMulation).

The first system, developed in co-operation with the 'Instituto Nacional de Tecnologia', Brazil was intended to be used in assessing the effect of variances in jobbing and batch production structures in a typical job-shop environment. This first system is described in appendix A. The experience gained in implementing the job-shop model was used as a basis for the development of ESSIM. Though the main design decisions had been taken prior to the start of the coding process, practical experience led to many revisions of the original plan. These are discussed in section 4.2.2.

ESSIM, in the form in which it presently stands, is designed on a modular basis. In section 4.3, the role of the individual modules and their interaction is explained.

The expert system module was specifically developed for the purpose of simulation modelling. The module can nevertheless be used on a stand-alone basis and has a number of novel design features that should be of interest to researchers working in AI. Section 4.4 describes the language syntax of the ES and explains how the facilities were implemented. Particular emphasis is placed on how the module accesses external variables and Pascal procedures and functions.

The discrete event simulation module of ESSIM is based on the three-phase world view. A library of Pascal routines used for teaching simulation at the LSE has been modified and extended to permit calls to the expert systems and sharing of data structures. Section 4.5 describes the development of the library and explains how the module interacts with the expert system.

The expert system and simulation modules must share data values using common variables and data structures. Replicating data would be inefficient in terms of speed and memory capacity, and would lead to the risk of inconsistencies in stored information. The communications interface linking the expert system and simulation module is consequently of critical importance. Section 4.6 explains the role of the expert system 'Communications Interface generator' (CI-generator) and describes how the variables are shared.

Simulation modelling and expert systems rely on effective communication of information. An individual creating a model must expend significant effort in the creation of a user-friendly interface and output displays for relaying critical data in an effective form. Microsoft Pascal in which ESSIM is implemented has no facilities for output control or graphics. A library of routines was consequently developed, based on state-of-the-art techniques including multi-level windowing, iconics, and mouse control. Though useful, the routines were found to be time consuming to use and the process of designing a screen display laborious. An additional module called DESIGNER was therefore created that permits screen designs to be created interactively and the corresponding code generated automatically. Section 4.7 describes the development of the man-machine interface routines and the role of DESIGNER in the development of a simulation model.

4.2 RESEARCH STAGES

Prior to the development of ESSIM, an experimental system was developed which was designed solely for job-shop scheduling problems (See Goodman et al.[1987]). The experience gained in developing this specific application was essential in helping to specify the required features of a generalised simulation environment encompassing characteristics of both simulation and AI. This conceptual design led to the creation of ESSIM.

4.2.1 Simulation of a job-shop.

The job-shop application was developed as a generalised model which could be adapted to different manufacturing environments. The recursive nature of product manufacturing in job-shops permitted the specification of a generalised assembly process based on a number of work centres and products. Each product is made up of a number of components which are in turn constructed from a number of sub-components. Components can either be assembled at work centres or brought in as raw material stock from external suppliers. Raw materials have to be re-ordered at intervals to ensure that bottlenecks are not created by the lack of a component. Each assembly process can only take place at specific work centres were the appropriate machinery is available. When an order is received for a quantity 'x' of a product, the manufacturing of the appropriate quantity of each of the sub-components is scheduled. Items assembled at work-centres are either placed in stock ready for delivery to a customer or transferred to another work-centre, ready for the next phase in an assembly process.

The specification of the products being manufactured, the characteristics of the jobs being performed, re-ordering procedures for raw materials, and details of orders received are specified interactively and stored in a database. The provision of a robust and flexible user interface was seen as being critical given the intended role of the system as a tool for decision support. The interface enables the modeller to alter entries in the database and modify features of the production process. Hence, the modeller can define the characteristics of the environment to be modelled and can carry out experiments without the need to modify the physical code.

One of the problems in experimenting with the job-shop model was the modellers ability at interpreting the generated model output. A graphics based interface was seen as the most appropriate means of permitting the depiction of the dynamic behaviour of the simulated job-shop. Queue build-ups and stock-levels can be scrutinised though high resolution graphs of a selected set of work centres. Other display formats include a diagrammatic representation of the job-shop depicting the flow of jobs between work centres. (see figures 12 and 13). Data can also be displayed in a textual format. For instance, details of orders completed and those still outstanding can be viewed on request either during or following the execution of the simulation program. Such facilities, give the modeller a view of the real-world and permit simple analysis of the relative performance of alternative model designs without recourse to an experienced modeller.

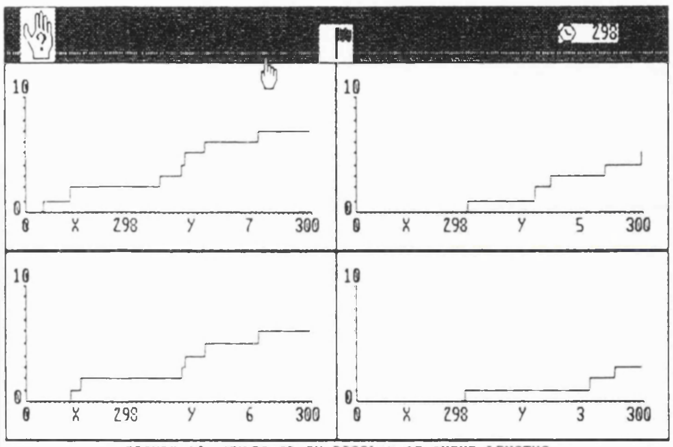


FIGURE 12 MULTI-GRAPH DISPLAY OF QUEUE LENGTHS

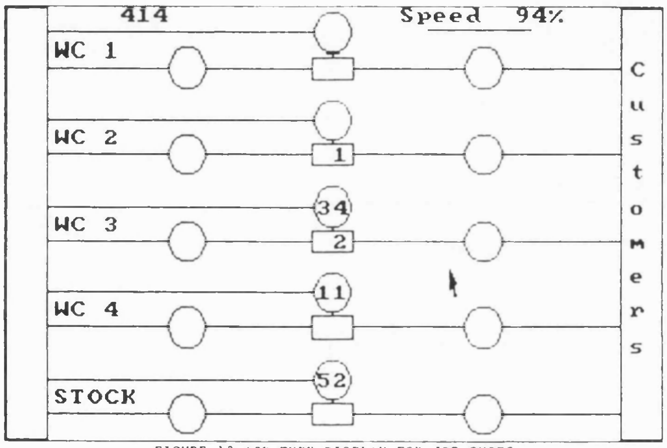


FIGURE 13 ACD TYPE DISPLAY FOR JOB SHOPS

One of the limitations of the job-shop modelling environment was the restrictions imposed on the range of experiments that could be carried out. Whereas it was possible to experiment with the addition of work-centres, or the manufacturing of new products and sub-components, no investigation was possible into the effect of alternative production strategies without recourse to a programmer. The concept of creating a separate knowledge-base allowing the modeller to alter and experiment with production related decision rules was therefore felt to be justified.

The addition of a rule based expert system to the existing job-shop application could not be achieved within the constrained development environment provided by the Turbo-Pascal compiler. The decision to use Turbo-Pascal for the first stage of the research was made on the basis that the language supported rapid prototyping. A comprehensive graphics library is provided, easing the development of the man-machine interface. Secondly, code compilation is completed in seconds rather than minutes. Turbo-Pascal was not appropriate for the later stages of the research because of the limit of 64K on code size. A language that would allow the separate compilation of modules into object code was required. The simulation model, expert system, and interface could then be developed independently and linked together to produce the final program. Microsoft Pascal fulfilled these requirements and had the benefit of permitting mixed language programming. Hence, library routines from the Microsoft C and Fortran compilers could be used. However, two major limitations had to be accepted. Compiling a single module and then linking the code could take up to 10 minutes. Secondly, no cursor control and graphics drawing facilities were provided. Translating the existing code from Turbo-Pascal was consequently a complex process necessitating both code modification and the creation of a library of graphics routines. Turbo-Pascal command names were retained enabling research students to upgrade their simulation programs to the Microsoft compiler.

Having implemented the job-shop model in Microsoft Pascal, the next stage of developing the expert system began. The first version of the inference engine was based on a simple forward chaining inference strategy. This enabled rapid development of the expert system and required only limited modification of the simulation model's three-phase executive. At each time advance, control would be passed to the expert system module and the current variable values used as input for the inference process (see figure 14). This first prototype was used as a basis for the development of the rule-base editor and the necessary routines that would enable the sharing of variables and data structures between the simulation and ES. Simple rules were used in testing the functionality of the system.

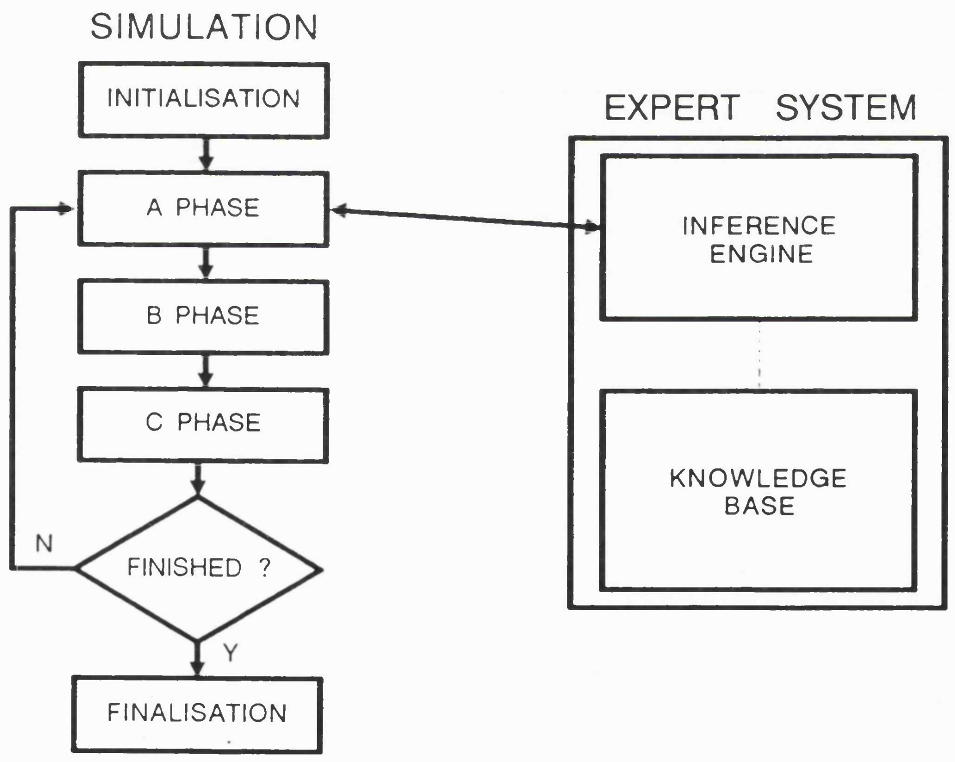


FIGURE 14 A FIRST PROTOTYPE SYSTEM

4.2.2 The development of ESSIM

Having established that a simulation model and expert system could be made to interact, the development of the ESSIM environment began. The functionality of the job-shop environment was to be retained whilst allowing the modeller to develop his own simulation application. One apparent problem was the considerable work that the programmer would have to undertake in implementing an effective man-machine interface. The relevant routines for menu selection, windowing and mouse control would be available in a library of routines but considerable expertise would be needed in making use of them. The problem was resolved by developing a separate program called DESIGNER which allows the modeller to develop the interface interactively and then automatically generate the corresponding program code. The mouse is used to position and scale the necessary windows and menus. DESIGNER can also be used for accessing pre-declared software modules such as the knowledge-base editor, and listing the content of output files in specified windows. (See appendix E.)

During the development of ESSIM, the expert system was re-written using a more complex backward chaining inference strategy. The syntax used in the knowledge-base was altered and extended. Facilities were then created for accessing Pascal procedures and functions. Though developing the expert system (ES) was by no means a simple problem, creating the interface between the simulation model and ES proved to be the most challenging and complex problem to solve. The difficulties mainly arose because of the difference in nature between the simulation and expert system. The simulation model is written using the Pascal language syntax and compiled. In contrast, the expert system is a development language, itself written in Pascal. The knowledge-base is interpreted and consequently cannot directly access compiled variables declared in the simulation model. The problem was overcome by developing a 'Communications Interface' (CI) between simulation model and expert system knowledge base. Requests for variable values are passed to the CI which returns the address at which the appropriate value is stored. Calls to procedures and functions from the knowledge-base are also handled by the CI.

When new variables are defined which are to be shared between expert system and simulation model, modifications have to be made to the CI. Such changes are handled automatically by the 'Communications Interface generator' (CI~generator) which scans the knowledge-base, and generates Pascal code which forms the link between the simulation model and ES data structures. The functionality of the CI and CI-generator is described in section 4.6.

The development of ESSIM also led to changes having to be made to the simulation module. The use of DESIGNER in creating the interface meant that output could no longer be written directly to the screen but had to be directed to specified windows. Altering the expert systems inference engine to a backward-chaining goal directed strategy also led to modifications of the simulation module. Routines had to be developed, enabling the ES to be activated by a call from the simulation module. Such calls, detail the goal to be solved and the location of the relevant rules. The simulation routines that establish the calls are also designed to return any result obtained from the inference process. The characteristics of the simulation module are discussed in detail in section 4.5.

4.3 OVERALL SYSTEM DESIGN

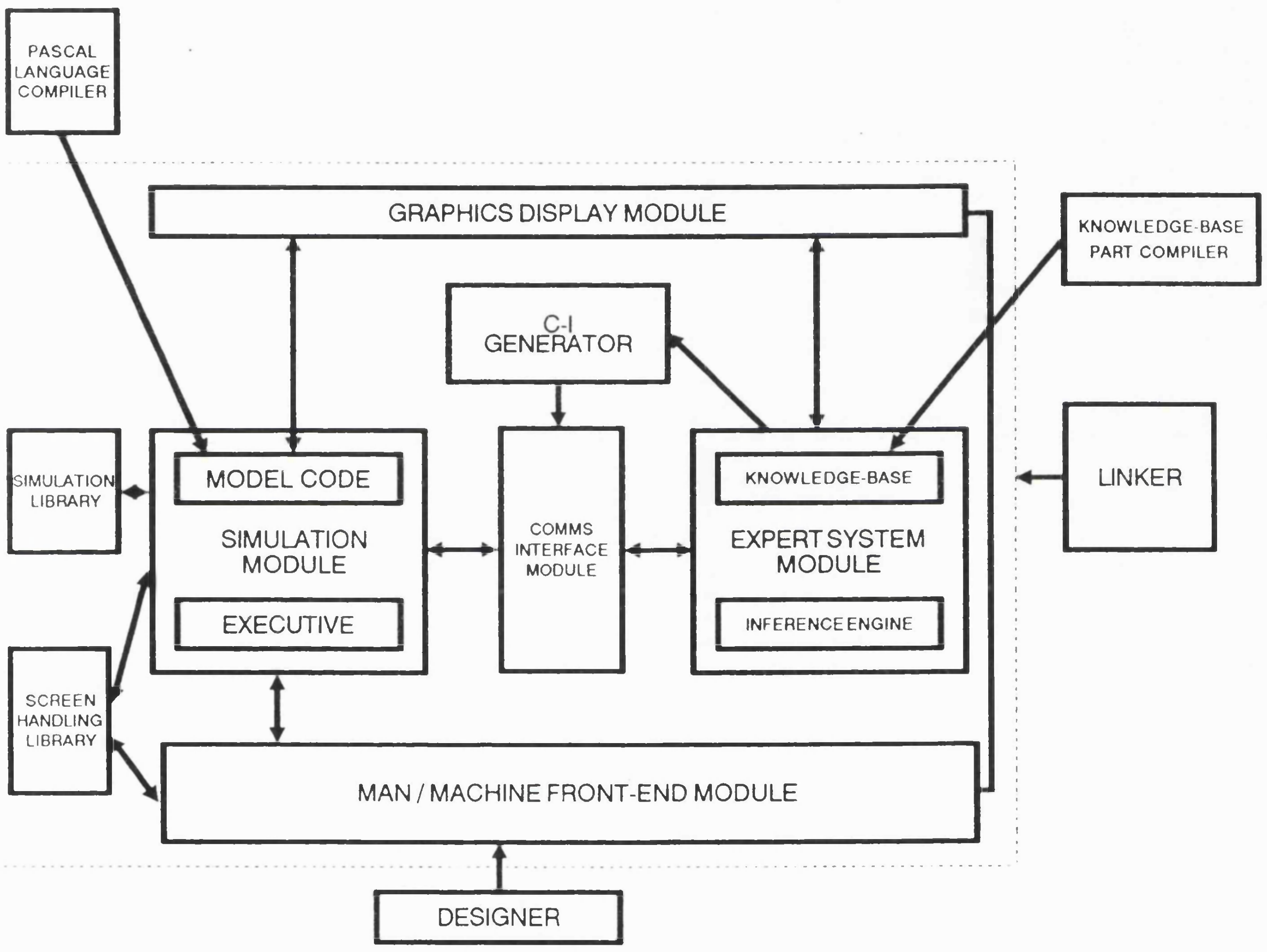


FIGURE 15 DESIGN OVERVIEW

Figure 15 is a diagrammatic representation of the ESSIM model development environment. As we have seen, the simulation module and expert system module interact via an intermediary communications interface module. The communications module is dependent on code generated by the

'CI-generator'. The simulation and expert system modules are both divided into two components. The simulation module consists of an 'executive' and a set of user-defined files which contains the formal definition of the model events. The executive maintains a diary of pre-scheduled events and controls the execution of the simulation. The user-defined model is written in Pascal and is compiled to object code. The expert system module consists of an inference engine and a user defined knowledge-base containing the decision rules that are used in the real-world. The content of the knowledge-base is translated into more compact and usable code by the 'knowledge-base part-compiler'. Part-compilation is necessary each time a change is made to the knowledge-base and immediately precedes invocation of the expert system. (The difference between a part-compiler and traditional compiler will be explained in section 4.4.3)

The 'graphic display module' (figure 15) consists of the necessary routines for the dynamic display of run-time output and for the creation and manipulation of windows. The module is controlled by calls from the simulation model and/or the expert system knowledge-base. Relevant data may be passed as parameters to the graphic display routines or accessed directly from the database of the communications interface module. The database contains references to all shared variables and data structures.

The 'man-machine front-end module' (figure 15) has a dual role. Initiating the simulation run and controlling the exchange of data between the end-user and relevant software. Requests for user input are displayed in windows and so there is a close association between the man-machine front-end and the graphics display module. Extensive use is made of mouse controlled multi-level menus for the selection of run-time options and for the activation of pre-defined routines. These routines control such aspects as the input of data into the knowledge-base editor and the invocation of DOS facilities such as directory listings.

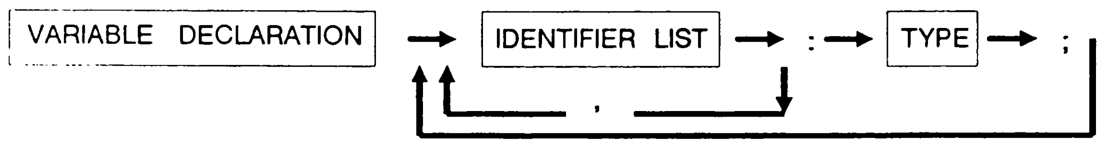
All the modules described were designed so as to be usable on a stand-alone basis. A simulation model can be developed without an expert system or vice-versa. The graphics display and man-machine interface routines can also be used with other Pascal programs. Giving the software engineer such flexibility may nevertheless give rise to problems. Difficulties could for instance be encountered in making use of the library routines or linking the relevant modules together. Two programs were consequently developed to help simplify the development process. A program generator called 'Designer' (appendix E) is used interactively in creating pull-down menus and windows for the display of text and graphs. The code produced corresponds to the required graphics display and man-machine front-end modules. The other program, named 'Linker' takes the modeller through the necessary steps in compiling and linking the relevant modules. 'Linker identifies the Pascal libraries that are needed in generating the final program from the object code modules and is also responsible for activating the CI-generator. The CI-generator scans the knowledge-base and generates code for the communications interface module which is then compiled.

4.4 DESIGN OF THE EXPERT SYSTEM COMPONENT

The characteristics of the expert system module will now be discussed by considering in turn the design of the knowledge-base and inference engine.

4.4.1 The knowledge-base

The expert system knowledge-base is an ASCII text file consisting of a declarations part and separately identifiable sets of production rules.
The knowledge-base can share variables with the simulation model or make use of local variables. The structure of the declarations is intentionally similar to the Pascal syntax. (See figure 16).



```
EXTERNAL     ExportsLeftForShip, NumbFreeBerths, Duration: INTEGER;
             _LoadShip: BOOLEAN;
             Count, Speed: REAL;

LOCAL        StartDockAtberth, StartShipLeave: BOOLEAN;
             ShipCode: CHAR;
             ContainerCode: LSTRING(20)
```

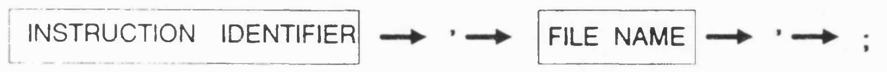
FIGURE 16 VARIABLE DECLARATIONS

The variable declaration 'EXTERNAL' is used to identify variables that are to be accessible by all modules. The corresponding identifier list must consist of variable names that have been, or will be declared as shared 'PUBLIC' variables in either the simulation module or any of the other modules being used.. The declaration 'LOCAL', denotes variable that can only be used by the expert system. The types that can be associated with the variables are identical to those available in Microsoft Pascal but exclude double precision and user defined types.

Local and external variables referenced in expressions and statements are fully compatible. However, the inference engine handles the values differently. External variables have an associated value, whether defined in the expert system or in another module. The only exception are external variables preceded by the symbol '_'. These are reset to being 'undefined' at the end of each call to the expert system. In contrast, local variables are initiated as being undefined and are set to given values by the execution of statements. Furthermore, local variables have their value reset to being 'unknown' at the end of each call to the expert system.

Pascal Procedures and Functions declared in an object code library or program module can be called from the expert system. (figure 17)

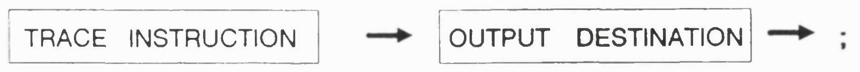
The file name must be that of a Microsoft Pascal module. The procedures and functions defined in the file can then be accessed from the expert system knowledge-base. The only restriction is that parameter passing cannot be used. The equivalent effect can nevertheless be achieved using shared variables. Parameter passing was omitted from the expert system specification because of the development overheads that would have been incurred.



PASCAL FILE 'Rules.pas' ;

FIGURE 17 DEFINING PASCAL PROCEDURES & FUNCTIONS

Verification of the rules defined in the knowledge-base is a complex problem because of difficulties in knowing exactly what the inference engine is going to do. The expert systems consequently has a trace facility that lists the individual steps taken during the inference process. (See figure 18).



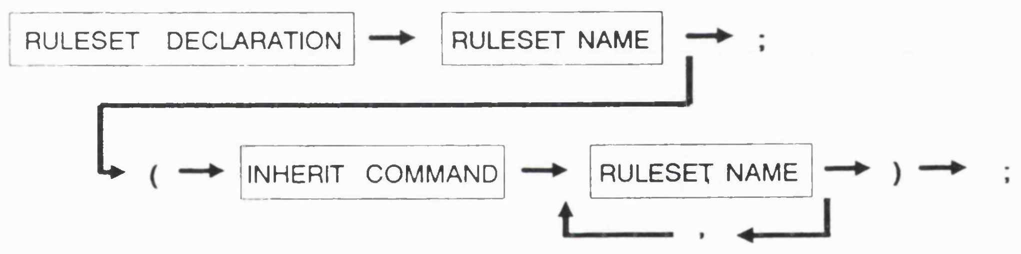
Example 1:	TRACE TO FILE 'Trace.txt';
Example 2:	TRACE TO HIRES 5,7,50,20;
Example 3:	TRACE TO TEXT WINDOW 2,3,70,15;
Example 4:	TRACE TO DESIGNER 9;

FIGURE 18 INFERENCE ENGINE TRACE FACILITY

The output produced by the trace can be directed to a number of destinations. If the screen display is already cluttered, output can be sent to a text file (example 1, figure 18). The text file can then be inspected by temporarily interrupting the simulation or by waiting until the end of the run.

The trace can alternatively be directed to a window of defined size in either graphics or text modes (Examples 1 and 2, figure 18). The content of the window scrolls as the information gets generated. Finally, compatibility with the DESIGNER interface generator is maintained by allowing output to be directed to any pre-defined window (example 4, figure 18).

The remainder of the knowledge-base consists of a number of 'rule-sets'. (see figure 19)

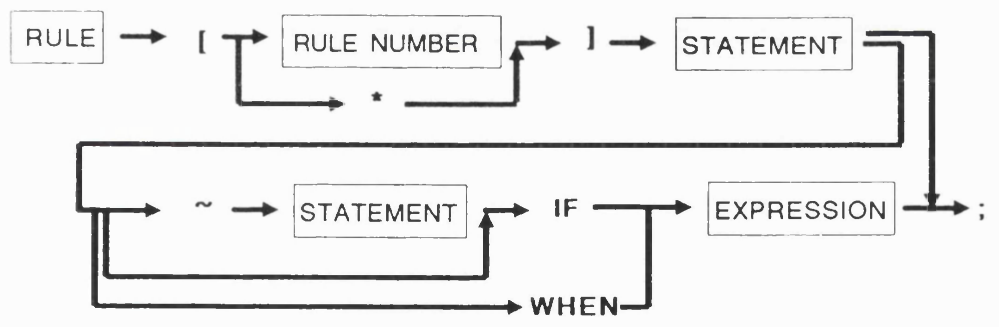


```
RULESET CraneManager (INHERIT ImvManager, ShipManager);
rule1
rule2
rule3
.
.
.
.


RULESET ShipManager;
rule1
rule2
rule3
.
.
.
.
```

FIGURE 19 RULESET DECLARATIONS

The rule-sets each consist of a number of production rules. Rule-sets typically identify part or all of the decision rules applied by individuals represented in the simulation model. The end of a rule-set is identified as being the start of the next rule-set thus eliminating the need for file markers. A rule-set can 'inherit' rules from another rule-set, thus alleviating the problem of rule replication. When a rule in a rule-set cannot be resolved, the inference engine checks for the presence of an 'INHERIT' statement. The rule-sets specified are then scanned in turn in attempting to satisfy the rule.

The syntax of the production rules is as follows: (Figure 20)



```
[*] NumberOfBerths = 2 ;  {Maximum Number of Berths is 2}
[1] ShipNumber = ShipCode ;
[2] ShipJobsLeft = True WHEN (ShipImportJobs > 0) OR (ShipExportJobs > 0);
[3] BerthedShip = True WHEN (ShipInBerthOne = True) OR (ShipInBerthTwo = True);
[4] ShipFullOfExports = True WHEN ExportsLeftShip = 0 ;
[5] _MovegantryToLandSide = True IF
     ((NumbMoveToShipJob * NumbMoveToExitJob > 0) AND (PriorityToShipJobs = False));
[6] (STARTSHIPARRIVE = True) AND (_Time = 1200) ~ (STARTSHIPARRIVE = False) IF
     (NumberOfShipsAtSea > 0) AND (ShipArrivalDue = True);
```

FIGURE 20 ESSIM KNOWLEDGE-BASE RULES

The expressions must yield a result of the standard type 'boolean'. If an expression produces the value 'True', then the statement defined as the first part of the rule is executed. The following rules of precedence identify the order in which operations are performed. (figure 21)

OPERATOR	PRECEDENCE
* / AND	1
+ - OR)	2
(	3
= <> < <= > >=	4

FIGURE 21 OPERATOR PRECEDENCE

The basic rules of precedence are applied as in standard Pascal. An operand which is located between two operators of different precedence is always bound to the operator of higher precedence. Secondly, if the operand is located between two operators of the same precedence, then the operand is bound to the operator situated to its left. Thirdly, expressions within parentheses are always evaluated first.

By using combinations of the boolean operators in the 'expression', rules of considerable complexity can be defined. Boolean operators other than 'OR' can also be used in the 'statement' part of the rule. There is no limit to the length of a rule.

Both local and external variables can have their values set by association with Pascal functions. Pascal functions can be used in either the expressions or statement part of rules. The Pascal functions can be used as a way of circumventing the problem of setting variables to values computed from, or stored in, files, arrays, and records. Procedure calls can be made by specifying the procedure name in the statement part of rules. Procedures can be used in specifying algorithms or as a means of executing low-level commands (e.g. writing to files, manipulating the display etc..).

As shown in rule 1 of figure 20, the condition section of a rule can be omitted. The statement is then executed unconditionally if the value of 'ShipNumber' is required during the inference process. As shown in the first rule of figure 20, the rule number can be replaced by an asterisk. The effect is two-fold. Firstly, the statement is executed on activation of the expert system. Secondly, the variable 'NumberOfBerths' defined in the statement retains its value until the end of the simulation run.

In rule 3 of figure 20, the reserved word 'WHEN' is used to separate the conditional statement from the 'action' part of the rule. If the conditional part of the rule is satisfied, the variable 'BerthedShip' is set to the value True. Conversely, if the condition statement cannot be satisfied the value of 'BerthedShip' is set to False. The reserved word 'WHEN' can only be used where a single boolean statement is used in the action part of a rule.

In rule 5 of figure 20, the reserved word 'IF' is used. If the conditional part of the rule is satisfied, the variable '_MoveGantryToLandSide' is set to True. However, if the conditional statement cannot be satisfied the action part of the rule does not execute.

In rule 6 of figure 20, the reserved word 'IF' is again used. The syntax is however slightly different from that of rule 5 in that the symbol '~' is used in the action statement. If the conditional part of the rule is satisfied, the statements preceding the '~' symbol are executed. If the conditional part of the rule cannot be satisfied, the variable 'StartShipArrive' that follows the '~' symbol is set to the value False.

4.4.2 Modelling "Cooperative Decision Making".

As we have seen in section 4.4.1, the concept of rule-sets was introduced into the ESSIM expert system in order to segregate the rules applied by each decision maker. The use of rule-sets permits decision rules to be grouped according to the name of the decision maker or by job function.

When resolving a goal, the expert system inference engine is limited to searching through the rules contained within a given rule-set. The rule-set to be used in resolving a goal is passed from the simulation model to the expert system as a parameter along with the details of the goal to be resolved. In many cases, the rules applied by a given decision maker are insufficient for the purpose of resolving the defined goal. In the real-world situation, operational policies are often enacted by more than one individual. (A concept that will be referred to in this thesis as "Cooperative decision making"). For instance, one individual may consult another or several individuals may work together in resolving a problem. In other cases, a line manager may always have a final veto over a decision taken by a more junior member of staff. In order to reflect the involvement of multiple decision makers in resolving an operational "Goal", the concept of inheritance was introduced. In the event that a goal cannot be resolved using the rules applied by a given decision maker, the expert system inference engine may consult one or more further rule-sets.

An interesting point about "Cooperative decision making" is that the grouping of rules is by decision maker and not by activity. In attempting to resolve a given goal, the search space may be extended to those rules used by decision makers involved in operations described by other activities in any of the activity cycles.

The use of rule-sets introduces a concept of rule modularity. This modularity can potentially be used to reflect simple hierarchical management structures. Consider the simple case where there are two rule-sets, one relating to a shop-floor operator and the other to a line manager. There are two ways in which the influences of management can be represented within the model. Firstly, the rule-set associated with the operator may reflect the operators inability to resolve certain problems. By using inheritance to link the two rule-sets, the manager's expertise or authority can be brought into play in resolving the goal. In this example, the rules associated with the operator are given a higher priority. The manager's rule-set is only used in the event that the goal cannot be resolved. The second way in which the influences of management can be reflected in the model is by defining a sub-goal within the operator's rule-set. Consider the case in which the operator is able to resolve a given problem but is unable to proceed with an action until authorisation has been sought. By introducing a sub-goal into the operator's rule-set, control can be transferred into the manager's rule-set. In this example, the rules applied by the manager have a higher priority than the rules associated with the operator. Although the top level goal may have been resolved by the operator, the manager's rules may be applied in overriding the operator's planned actions.

4.4.3 The knowledge-base part-compiler

The part-compiler is activated prior to the start of the simulation run. The responsibilities of the part-compiler include memory management for the defined variables, the handling of calls to function and procedures, the optimisation of the knowledge-base code, and the detection of syntax errors. Whereas these are the traditional functions of a language compiler, the rules remain interpreted and hence the term 'Part-compiler'.

1.Memory management: The first step taken by the part-compiler is to extract references in the knowledge-base to local and external variables. The variable names, associated type and memory location, are then added to a tree using the following record structure. (Figure 22)

```
PointerToVarNode = ^VarNode ;

VarNode = RECORD
              Lower_Branch, Upper_Branch : PointerToVarNode ;
            VarName : String ;
            VarType : CodeNum ;
            VarAddr : ADS OF Byte ;
            Local : Boolean ;
            Undefined : Boolean ;
            Perm : Boolean ;
          END;
```

FIGURE 22 RECORD DECLARATION FOR THE VARIABLE TREE

The boolean variable 'Local' specifies whether the 'VarName' variable is specific to the expert system or whether its value is defined as a public variable in Pascal and consequently shareable with the simulation module.

All variables defined as being local in the declaration part of the knowledge-base have no associated value at the start of the inference process. Variables defined as 'External' in the knowledge-base may have had their values set in the simulation module and consequently the 'Undefined' field is set to False. The only exception are 'External' variables which start with the underscore character '_'. Such variables are initially set to the 'Undefined' status. If the conditional part of a rule contains an 'Undefined' variable, the rule cannot be executed until a value has been associated with the variable.

At the end of an inference process, local variables have their status reset to 'Undefined'. The 'Perm' field is used to identify those local variables which should retain their value throughout the duration of the simulation run. In figure 20, the first rule has neither a conditional clause nor an associated rule number but the asterisk character instead. This indicates that the declared variables should have their values set by the part_compiler and that the 'Perm' field should be set to True.

'VarAddr' is the address of the memory location containing the first byte of the variable value. In the case of local variables, a memory request is made for RAM space corresponding to the number of bytes needed in storing the variable. The address of the allocated memory is then placed in the tree. If the variable is external to the expert system module, the memory space needed in storing the value is allocated by the module in which the variable is declared.

The memory space is only allocated at run-time and the location of this memory space will differ each time the module is executed. The problem of finding the variable address is further aggravated by the fact that the modules are compiled whereas the knowledge-base is interpreted. Identifying the relevant addresses can only be achieved from within a compiled routine linked to the relevant module. This routine is located within the 'Communications Interface' (CI). The creation of the CI is undertaken by the CI-generator, the functionality of which is described in section 4.6.1. The CI returns the run-time address of the external variables which are then added to the knowledge-base variable tree. During the execution of the expert system, references to a given variable value is achieved by scanning the tree for the relevant variable name and directly accessing the content of the corresponding memory address. Knowing the address of the variable is not however sufficient to extract the associated value. The Type of the variable is also required. The 'VarType' field is a one byte code identifying the type of the variable. The variable type determines the format used for the storage of the value at the specified address.

2.Function and Procedure calls: One of the difficulties in developing the inference engine was the implementation of the facility allowing calls to compiled Pascal or C functions from within the knowledge-base. The ability to incorporate compiled 3GL code into the otherwise interpreted expert system rule-set, permits the creation and use of complex algorithms.

Pascal, in which the expert system is written, does not support calls to procedures or functions using identifier names stored as text strings. The problem was overcome by treating the compiled functions in a similar way to the expert system variables. Prior to the execution of the expert system, the memory addresses of all procedures and functions contained in the file defined by the 'Pascal File' instruction (figure 22) are added to the variable tree. The type code (VarType) used in the variable tree record structure (figure 22) identifies the addresses as being related to compiled code.

At run time, the procedure or function addresses are used to transfer control from the expert system to the compiled routine. Pascal does not support function calls using addresses, though this facility is available is C. The problem was overcome by taking advantage of the mixed-language programming facility available with Microsoft compilers. At run-time, the inference engine passes the address of the relevant code to a C function contained in a compiled module. In turn, the C function passes control to the code stored at the specified address. When this segment finishes executing, control returns to the inference engine via the C function.

3.Code optimisation: The part-compiler converts the rules contained in the rule-sets into a structure optimised for code interpretation. The first step is to check for INHERIT commands. The list of rule-sets names defined as parameters to INHERIT instructions are then placed in record lists. These lists are attached to the nodes of a tree indexed according to the names of all rule-sets. The records structures are: (Figure 23)

The functionality of the INHERIT command is further discussed in section 4.4.4.

```
PointerToVarNode = ^InheritTreeNode ;
PointerToList = ^InheritListNode ;

InheritTreeNode= RECORD
                   Lower_Branch, Upper_Branch : PointerToNode ;
                   SideBranch : PointerToList ;
                   RuleSetNum : Integer ;
                 END;

InheritListNode = RECORD
                   Side_Branch : PointerToList ;
                   RuleSetNum : Integer ;
                 END;
```

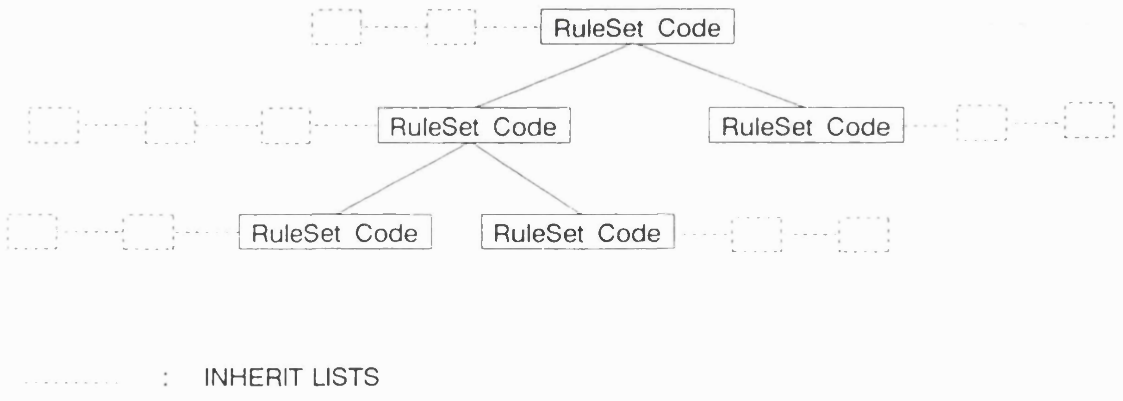


FIGURE 23 RECORD DECLARATION FOR THE RULESET TREE

The part-compiler then scans the content of each rule-set and creates a corresponding number of 'compiled' files. These temporary files reside on the disk and contain the optimised code used during the inferences. The fact that the files reside on a permanent storage medium, thus limiting the need for RAM, means that the potential size of the knowledge-base is only limited by the capacity of the disk. If execution speed is found to be unsatisfactory, and RAM space is available above the 640K DOS limit, a virtual drive can be used to remove the need for physical 'reads and writes' to and from disk.

The part-compiler creates the 'compiled' files by scanning and optimising the rules contained in the rule-set. The rules are in-turn read from disk and copied into a linked list. (Figure 24)

```
PntrToBuffer = ^RuleBuffer ;

RuleBuffer = RECORD
                RuleTxt : LineOfCode ;
                AddrNextLine : PntrToBuffer ;
             END;
```

FIGURE 24 LINKED LIST STRUCTURE FOR MEMORY RESIDENT RULES

If a rule is unconditional, a check is made to see whether an asterisk precedes the statement. The asterisk indicates that the statement should not be added to the rule file but should be executed immediately and that the status of any variables used should be set to 'permanent'. (See figure 20).

The rules which are in turn added to the linked list are optimised by replacing the IF or WHEN condition, Boolean operators, and dual character operators (<= , >= , <>) by single byte instructions. Statements and expressions defined in infix notation are then translated into 'Reverse Polish' ready for execution. Variables are replaced by two-byte codes. (See figure 25).

INFIX NOTATION:	JobDuration = ((10 + 5) / Total + WorkTime) * 17 IF (JobLeft >= 10 + SpareJobs) AND (CraneAvail = True);
REVERSE POLISH:	JobDuration 10 5 + Total / WorkTime + 17 * = IF JobsLeft >= 10 SpareJobs + AND CraneAvail = True ;
OPTIMISED ESSIM CODE:	$\underbrace{\text{AA}}$10 5+AB/AC+17*=AD#10AE+AF=1!

Result variable 'IF' Code End of line marker

FIGURE 25 THE PART-COMPILATION OF ESSIM RULES

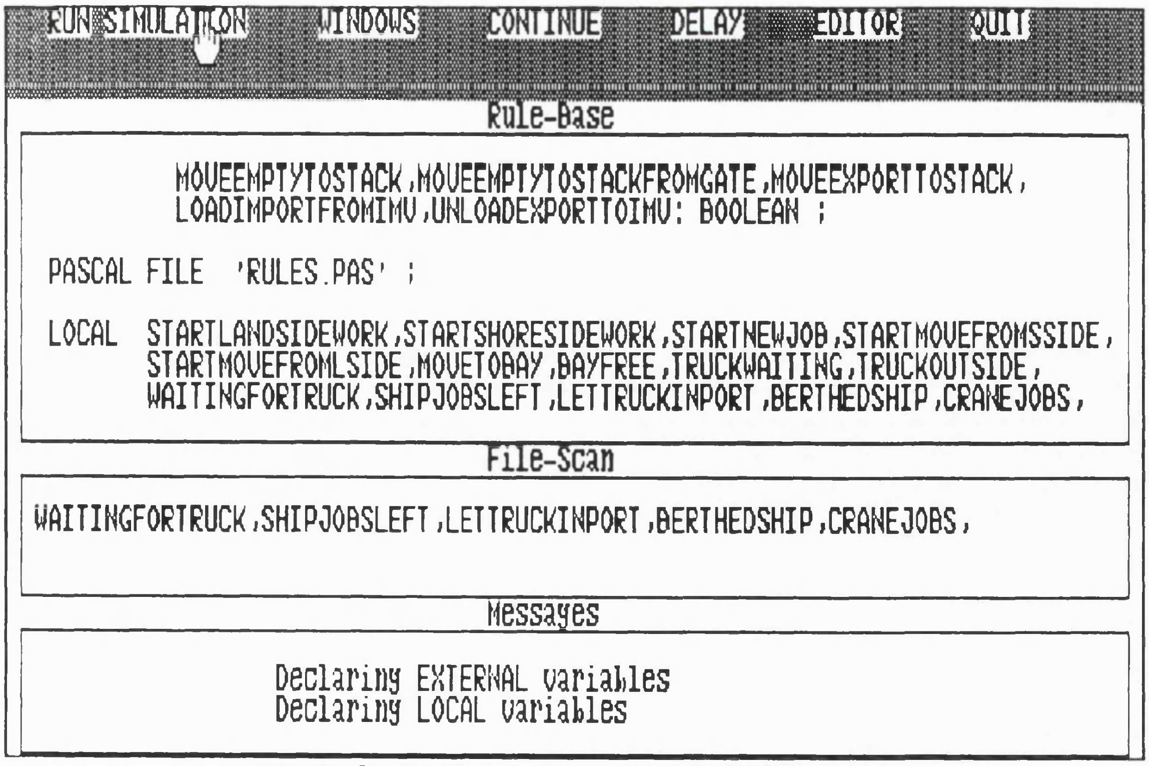
The knowledge-base part-compiler applies Dijkstra's method for translating from infix notation to reverse polish. An example is given in figure 26, based on the rule described in figure 25. A string and Last-In-First-Out (LIFO) stack are used, and the operator precedence defined in figure 21 is applied.

ORIGINAL STRING: JobDuration = ((10 + 5) / Total + WorkTime) * 17

STRING	STACK
10	(
10	((
10 5	((+
10 5 +	((+
10 5 +	(
10 5 + Total	(/
10 5 + Total /	(/
10 5 + Total / WorkTime	(+
10 5 + Total / WorkTime +	(+
10 5 + Total / WorkTime +	*
10 5 + Total / WorkTime + 17	*
10 5 + Total / WorkTime + 17 *	

FIGURE 26 DIJKSTRA'S METHOD FOR TRANSLATING TO REVERSE POLISH

4. Error detection & recovery: The knowledge-base part-compiler is responsible for the detection of syntax errors and inconsistencies in the variable declarations. The part-compiler produces a screen-based trace, as shown in figure 27. The upper-most window scrolls during the compilation process, displaying the content of the knowledge-base. If an error occurs, the modeller is aware that the problem manifested itself during the compilation of the last line of code shown in the window. The centre window is used to display the variables that have been extracted from the knowledge-base and added to the variable tree. The lower window is used for the display of user messages including explanation of errors. More complete diagnostics can be obtained by specifying the 'Trace To File' command in the knowledge-base (See section 4.4.1).



```
          MOVEEMPTYTOSTACK,MOVEEMPTYTOSTACKFROMGATE,MOVEEXPORTTOSTACK,
          LOADIMPORTFROMIMU,UNLOADEXPORTTOIMU: BOOLEAN ;

PASCAL FILE  'RULES.PAS' ;

LOCAL   STARTLANDSIDEWORK,STARTSHORESIDEWORK,STARTNEWJOB,STARTMOVEFROMSSIDE,
        STARTMOVEFROMLSIDE,MOVETOBAY,BAYFREE,TRUCKWAITING,TRUCKOUTSIDE,
        WAITINGFORTRUCK,SHIPJOBSLEFT,LETTRUCKINPORT,BERTHEDSHIP,CRANEJOBS,
```

```
WAITINGFORTRUCK,SHIPJOBSLEFT,LETTRUCKINPORT,BERTHEDSHIP,CRANEJOBS,
```

```
            Declaring EXTERNAL variables
            Declaring LOCAL variables
```

FIGURE 27 PART-COMPILER SCREEN BASED TRACE

Verifying the validity of the variables defined in the knowledge-base is achieved by checking for duplicate entries in the tree of variable names and corresponding memory addresses (see figure 22). The use of reserve words is also detected. Other errors involving variable names are harder to isolate. For instance, variables declared as 'external' in the knowledge-base must also be defined as being 'public' in one of the Pascal modules. Incorrect spelling leads to an inconsistency that cannot be detected by the part-compiler. The error is instead detected by the 'Linker' (see section 4.8) which checks for discrepancies between modules.

Syntax errors such as the omission of rule numbers and end of line markers are detected and reported by the part-compiler. The invalid use of variables is also easily identified. Some other errors may not be reported by the part-compiler but may nevertheless be identified visually from a trace produced during the variable declaration and rule optimisation processes. The part-compiler is also responsible for establishing the run-time trace by creating the necessary output file or window calls.

4.4.4 The inference engine

Calls to the expert system are initiated by the invocation of a function from any Pascal module. The goal to be resolved and the rule-set to be used are passed as parameters to the function. (Figure 28)

The GOAL function is part of the expert system inference engine. In figure 28, the rules declared in the 'CraneManager' rule-set will be used in trying to solve the goal 'CraneJobs'. The goal is to identify whether there are jobs for the crane to carry out and which of these should take priority. The GOAL function returns the address of the result. Consequently, the programmer must be aware of the data format used for the storage of the result. The benefit of using an address is that the function can return any data type. The address returned by the GOAL function is not the only way of returning a result. Any number of shared variables can be used for the same purpose.

$$\begin{tikzcd}
\boxed{\text{RESULT ADDRESS}} \arrow[r] & \text{:= GOAL('} \arrow[r] & \boxed{\text{RULESET NAME}} \arrow[r] & \text{','} \arrow[r] & \boxed{\text{GOAL NAME}} \arrow[r] & \text{';}
\end{tikzcd}$$

```
ResAddr := GOAL('CRANEMANAGER' , 'CRANEJOBS');
```

FIGURE 28 SYNTAX OF CALLS TO THE EXPERT SYSTEM

Having received the relevant instructions, the inference engine opens the 'compiled' file corresponding to the rule-set. External variables commencing with the underscore character and all local variables are then initialised to the 'undefined' status (figure 22). The backward chaining strategy used by the expert system makes use of a stack in directing the inference process. The first record of the stack contains a reference to the goal to be solved. Subsequent records list the sub-goals that need to be considered before a solution can be found for the main goal. The record structure is shown in figure 29.

```
RecPointer = ^GoalSearch ;

GoalSearch = RECORD
                NextRec : RecPointer ;
                LastRec : RecPointer ;
               Goal : SymbolSting ;
               FileNumber : Integer ;
               Line : Integer ;
             END;
```

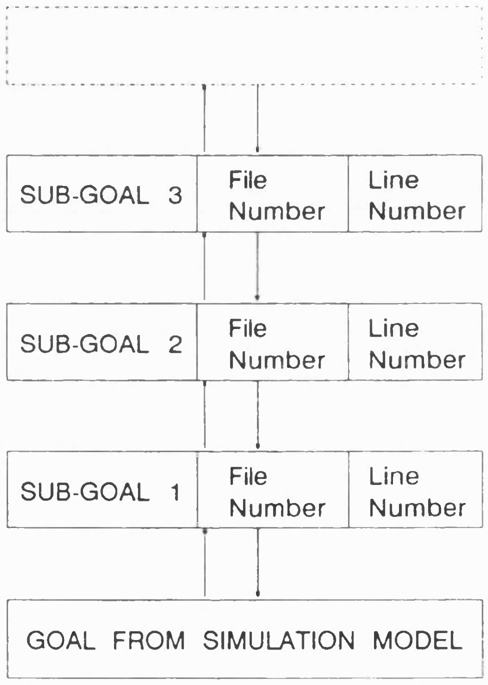


FIGURE 29 THE INFERENCE ENGINE GOAL CALL STACK

Let us now consider the example in figure 28. At the start of the inference process, the goal 'CraneJobs' is entered as the first record of the LIFO stack. The 'compiled' file corresponding to the 'CraneManager' rule-set is then scanned for an occurrence of the goal in the action part of one of the rules. If a rule is located that could potentially resolve the goal, the reverse polish condition expression is evaluated. Consider the following rule as an example: (Figure 30)

INFIX NOTATION:

```
[1] CraneJobs = True IF (Crane = 'Idle') AND
                              (JobDuration + 0.5 <= (DayEnd - TimeNow) * 1.1 + 1) ;
```

REVERSE POLISH NOTATION:

```
[1] CraneJobs True = IF Crane 'Idle' =  AND
                              JobDuration 0.5 + <= DayEnd TimeNow - 1.1 * 1 + ;
```

FIGURE 30 REVERSE POLISH RULE NOTATION

When searching through the knowledge-base, rules are loaded in turn into a 'linked list' data structure. The linked list permits the expert system to handle rules of an indefinite length. The linked list is also used as a temporary storage area for intermediate results. For instance, when evaluating a rules conditional expression, the constituent statements are evaluated and replaced by the corresponding boolean results (see figure 31). The conditional expression, now consisting uniquely of boolean values and operators can be resolved.

If the rule condition is satisfied, the goal has been resolved and control returns to the calling module. If the rule condition returns a boolean 'False', the rule-set file is searched for another occurrence of a rule statement that could be used to resolve the goal. If no rules can be found that will satisfy a goal or sub-goal, the files identified in the INHERIT list are searched in the order in which they are listed. If the GOAL cannot be resolved, a warning message is displayed on the bottom line of the display. The modeller then has the option of interrupting the simulation and updating the knowledge-base or ignoring the warning.

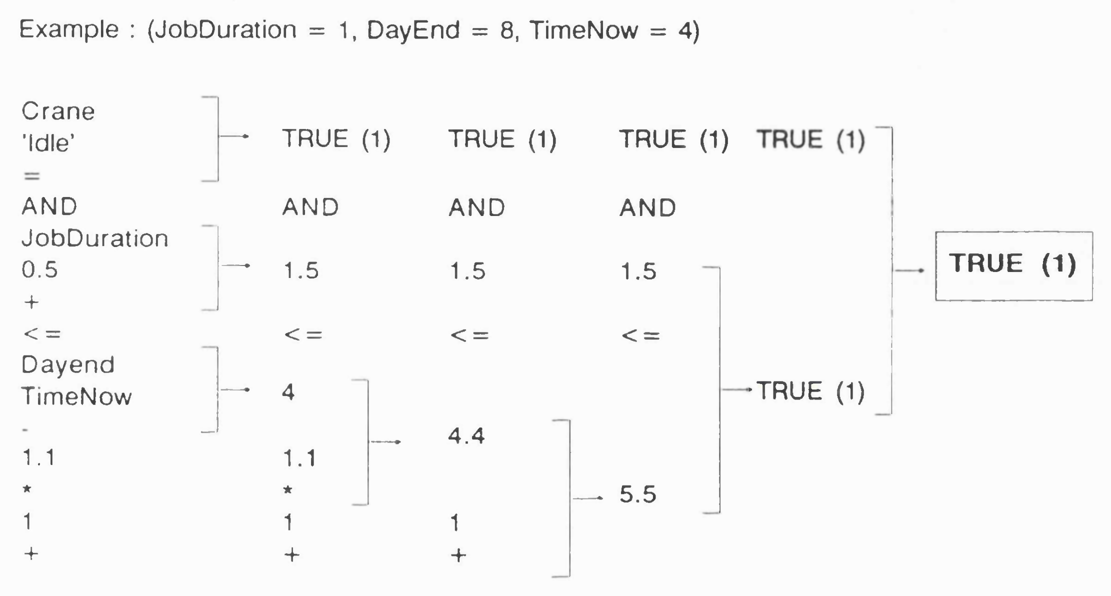
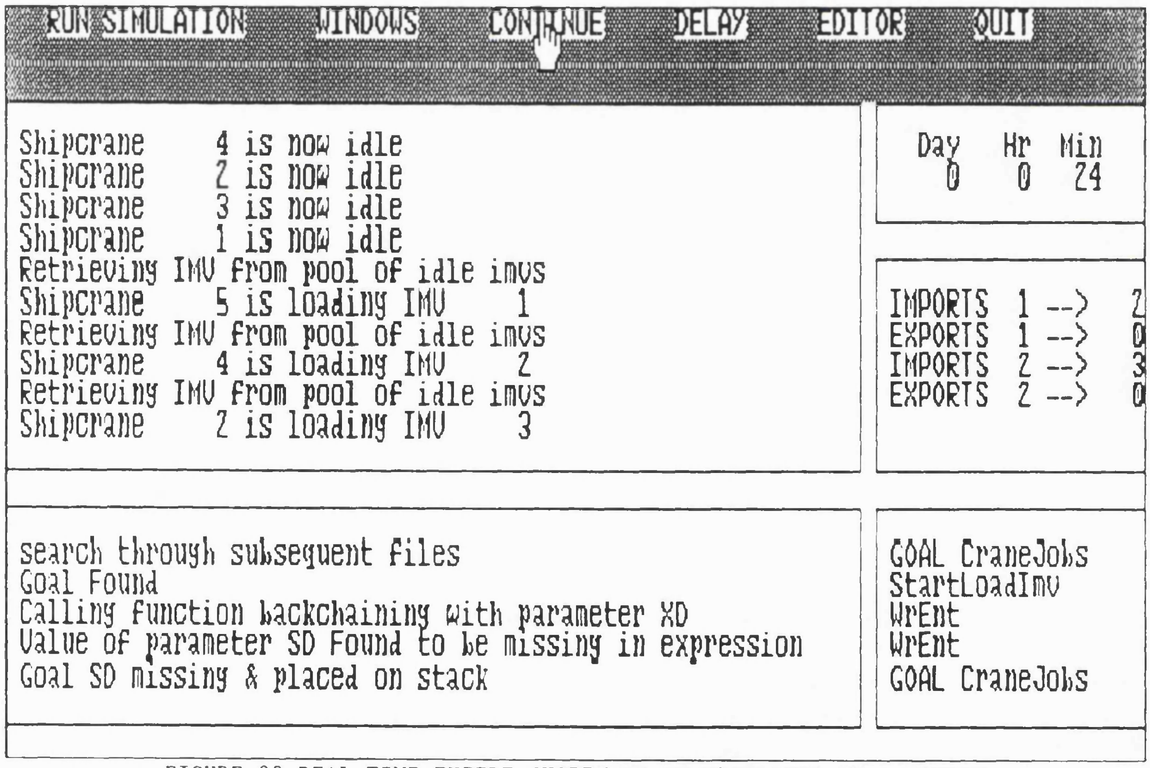


FIGURE 31 EVALUATING REVERSE POLISH RULES

If one of the local variables used in the conditional expression does not have a value, then the location of the rule is appended to the last record in the stack. A new record is then added to the stack with the variable set as a sub-goal. If a solution is found to the sub-goal, the last record in the stack is removed. A second attempt is then made at solving the previous goal, starting with the rule found at the location specified in the record. A solution has been found to the main goal when there are no longer any records left in the stack. The solution is then returned to the calling module as an address.

The logic applied by the inference engine can be scrutinised at the end of the simulation run by listing the content of a text file declared in the knowledge-base using the TRACE command (see figure 18). Alternatively, the same information can be displayed interactively in a scrolling window during the simulation run (figure 32). The trace is also invaluable in isolating logical errors in the rules.



```
Shipcrane     4 is now idle
Shipcrane     2 is now idle
Shipcrane     3 is now idle
Shipcrane     1 is now idle
Retrieving IMV from pool of idle imvs
Shipcrane     5 is loading IMV     1
Retrieving IMV from pool of idle imvs
Shipcrane     4 is loading IMV     2
Retrieving IMV from pool of idle imvs
Shipcrane     2 is loading IMV     3
```

```
Day   Hr   Min
 0     0    24
```

```
IMPORTS  1 -->  2
EXPORTS  1 -->  0
IMPORTS  2 -->  3
EXPORTS  2 -->  0
```

```
search through subsequent files
Goal Found
Calling function backchaining with parameter XD
Value of parameter SD Found to be missing in expression
Goal SD missing & placed on stack
```

```
GOAL CraneJobs
StartLoadImv
WrEnt
WrEnt
GOAL CraneJobs
```

FIGURE 32 REAL-TIME EXPERT SYSTEM TRACE DISPLAYED IN A WINDOW

4.5 DESIGN OF THE SIMULATION COMPONENT

The simulation model is written in Pascal using procedures and functions provided in a library of routines. The form of modelling used is based on the discrete event three phase approach due to Tocher[1962]. Changes in state in discrete event models take place at time intervals referred to as events. The three phase world view provides a framework for defining model dynamics in terms of events which may be categorised as being either time or state dependent. Model execution is controlled through a three phase executive which performs a time-advance in the A phase, executes all current time-dependent events in the B phase and examines and executes where appropriate all state-dependent events in the C phase (See figure 4). These three phases are well represented using the Pascal programming language because of the reliance on modularity in specifying model dynamics.

The suite of Pascal routines used in the simulation module are a modification of routines which were used for teaching simulation at the LSE. The routines, known as eLSE (Extended Lancaster Simulation Environment) are themselves a modification of Pascal routines developed at Lancaster University.

The first stage in the development of the simulation module was to transfer the eLSE routines to run under Microsoft Pascal. Students at the LSE were using the eLSE routines under Turbo-Pascal V.3, which imposes an unacceptable barrier of 64K on code size. Some modifications were necessary since Turbo-Pascal does not fully abide to the ANSI/IEEE standard (IEEE [1984]). Conversely, Microsoft Pascal does not support the use of graphics. Consequently, the necessary low-level routines had to be written to duplicate the functionality of Turbo-Pascal. The size of the eLSE routines was then reduced by removing unnecessary display related code.

The eLSE routines have been used in creating relatively small experimental models based on imagined real-world environments. Experience gained in developing the job-shop model showed that the use of an expert system would be of little practical benefit unless the simulation was sufficiently detailed as to warrant the creation of a separate knowledge-base. The use of a compiler supporting the creation of substantive programs was seen as a prerequisite to the development of simulation models under ESSIM. The creation of large models also requires the use of modular development techniques in keeping the code manageable and maintainable.

The existing eLSE routines impose structure on the modelling process by providing a model executive (the A phase) which controls the calls to the B and C phase procedures. The model framework assumes a single logical file containing all B and C routines, with the eLSE specific code stored separately within an 'include' file. In order to use the eLSE routines in a modular programming environment, the B and C phase procedures have to be appropriately grouped in modules. Interfaces then have to be created, allowing procedure and function calls between the separately compiled files and permitting the sharing of common data.

The eLSE routines make use of the CAUSE procedure to schedule the execution of a B phase event. The syntax of the procedure call is shown in figure 33. The integer parameter 'nb' identifies the B event procedure to be executed after a delay of 't' time units. During the execution of the A phase, (time advance) the scheduled B-events are activated through a call to the 'Call_For_Next_B_Event' procedure. A CASE statement is used to map between the procedure code and the event name (See figure 33).

SYNTAX OF B-EVENT CALLS USING ELSE

```
CAUSE(Nb: Integer; Ent: EName; T: Integer);


PROCEDURE   Call_For_Next_B_Event;
Begin
     CASE No_NextB OF
          1: B1;
          2: B2;
          3: B3;
          4: B4;
          ...........
          End;
End;
```

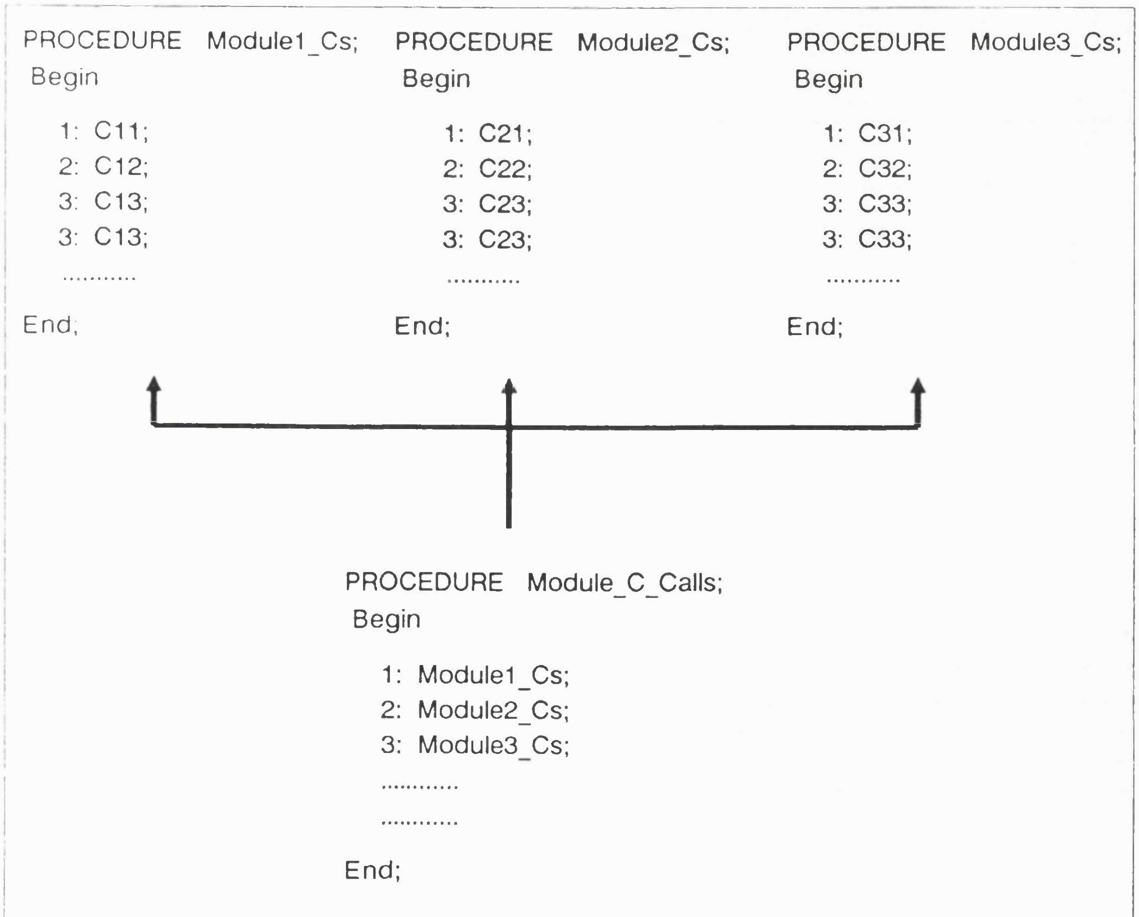
SYNTAX OF B-EVENT CALLS USING ESSIM

```
SCHEDULE(B_EventName: Address; Ent: EName; T: Integer);
```

FIGURE 33   SIMULATION MODEL B-EVENT CALLS

The eLSE routines required the definition of a procedure called 'Call_For_Next_B_Event' which contained the names of all B-events. This procedure had to be updated each time new B-events were added to the model. A simple modification was made to ESSIM which elimitated the need for the 'Call_For_Next_B_Event' routine. In ESSIM, descriptive names can be given to B-event routines. Instead of using the 'Nb' parameter to the CAUSE procedure, ESSIM expects to be passed the start address of the B-event procedure. During the execution of the A phase, the physical address of the B event is used to activate the procedure.

The eLSE environment makes use of the 'Go_Thru_C_Events' routine in activating the models C-event procedures (See figure 34). In ESSIM, a calling procedure for C-events is located in each code module (see figure 35). If a C-event is added to a module, then the modeller simply alters the calling procedure located in that module. Such a structure eliminates the need to re-compile multiple segments of code.



```
PROCEDURE  Module1_Cs;    PROCEDURE  Module2_Cs;    PROCEDURE  Module3_Cs;
 Begin                     Begin                     Begin

    1: C11;                   1: C21;                   1: C31;
    2: C12;                   2: C22;                   2: C32;
    3: C13;                   3: C23;                   3: C33;
    3: C13;                   3: C23;                   3: C33;
    ...........               ...........               ...........

End;                      End;                      End;
```

```
PROCEDURE  Module_C_Calls;
 Begin

   1: Module1_Cs;
   2: Module2_Cs;
   3: Module3_Cs;
   ............
   ............

End;
```

FIGURE 35   C -EVENT CALLS IN ESSIM

```
PROCEDURE Go_Thru_C_Events;
 Begin

           C1;
           C2;
           C3;
           C4;
           ...........

 End;
```

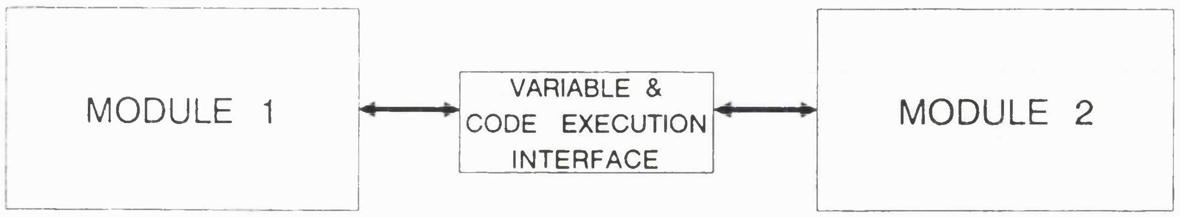
FIGURE 34 C-EVENT CALLS USING ELSE

The use of modules in writing simulation code necessitates some caution in the management of variables. The program design should reflect the natural modularity of a simulation model, depicted by its constituent activity cycles (Hills [1971]). Variables and associated data structures should then, as far as possible, be declared locally to each module. This minimised the size of the resulting program and improves the maintainability of the code.

The sharing of data between modules and the expert system should preferably be effected through intermediary interfaces (see figure 36). A programmer may for instance modify the data structures used in one module and neglect to reflect these modifications in other modules. The use of interfaces also forces the programmers developing the system to formally define the data accessible to individuals in the real-world. Consequently, the programmer(s) coding each of the modules only have access to data specified in the corresponding interface. An activity common to two activity cycles must be placed in just one of the corresponding code modules.

MODULE 1
MODULE 2

If changes are made to variables in module 1, corresponding alternations are required in module 2.



If changes are made to variables in module 1, changes are required to the interface, but not necessarily module 2.

FIGURE 36 INTERFACING MODULES

The need to develop the expert system knowledge-base in parallel with the simulation code significantly alters the development process usually associated with modelling. The disciplines required are described in chapter five specifically in the context of the development of the port model.

4.6 DESIGN OF THE COMMUNICATIONS INTERFACE

The responsibilities of the communications interface include the management of procedure and function calls from the expert system, and the sharing of data between the expert system and other modules.

The communications interface provides the necessary links between the compiled simulation code and the interpreted expert system. The incompatibility between compiled and interpreted code arises from the string oriented nature of the interpreter. For instance, the expert system scans the knowledge-base and finds that in order to solve the desired goal, a call has to be made to a function called 'JobDuration'. The function name has been extracted from the knowledge-base as a string and is consequently not recognisable by the Pascal compiler as a function name. The problem could be overcome by finding the start address of the function and then transferring control using the C routines which have been written (see section 4.4.3). Unfortunately, the address of a function cannot be identified unless the function name is 'hard-coded' in a compiled file. Using a Microsoft Pascal routine for determining the address of 'JobDuration' results in a return value corresponding to the address of the string and not the function. The same problem applies in making procedure calls and in determining the address of variables declared in a compiled module.

A second problem in linking the interpreted expert system with other compiled modules results from the syntax requirements of the Microsoft Pascal compiler. For instance, a procedure defined in one module can be called from another module provided that the procedure name is declared in the calling module as being 'external'. The same requirement exists for function calls and for sharing a variable between modules. Consequently, finding the start address of a procedure is not in itself sufficient for transferring control to that procedure. The procedure name must also be defined at the top of the module as being 'external'.

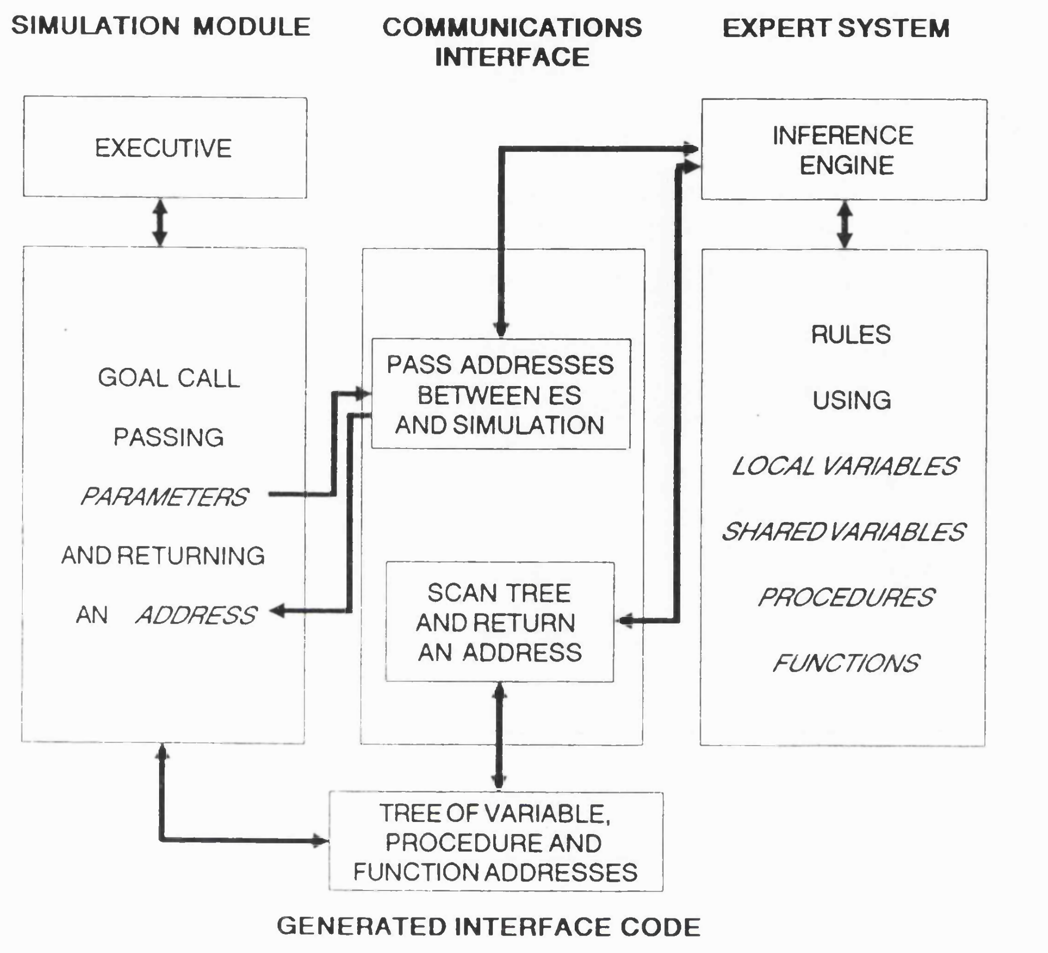


FIGURE 37 INTERFACE BETWEEN SIMULATION MODEL AND EXPERT SYSTEM

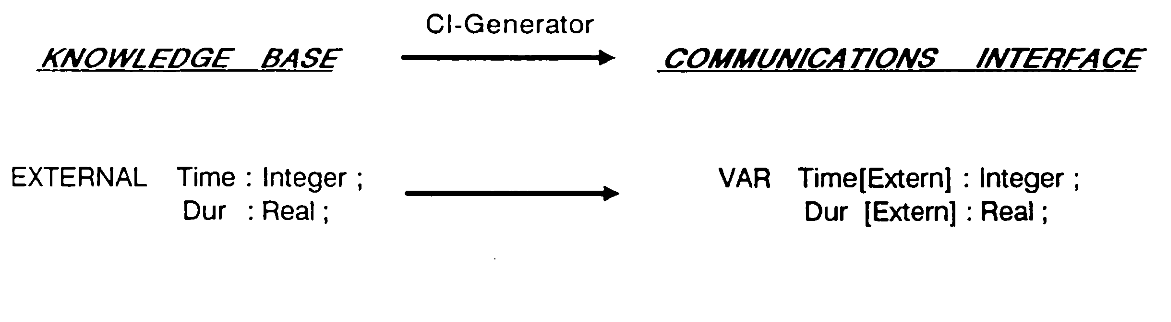
Short of writing ones own Pascal compiler, the problem of creating the interface could not easily be overcome. A possibility that eventually proved to be the most satisfactory was to develop a program generator that would create the communications interface module. The communications interface module contains all the necessary 'inter-module' declarations and returns the addresses of procedures, functions, or variables when passed to the interface as text strings (see figure 37).

4.6.1 The CI-generator

The CI-generator was developed as a means of creating the communications interface module. The CI-generator is activated by the 'Linker' program (see section 4.8) and need only be executed when additional procedures, functions, or external variables are added to the declarations part of a given knowledge-base.

When the CI-generator is activated, a template file is created using the same file name as the knowledge-base but with a different file name extension. The generator then adds code to the template file by directing output to 'Include' files. The generated code can be classified as follows:

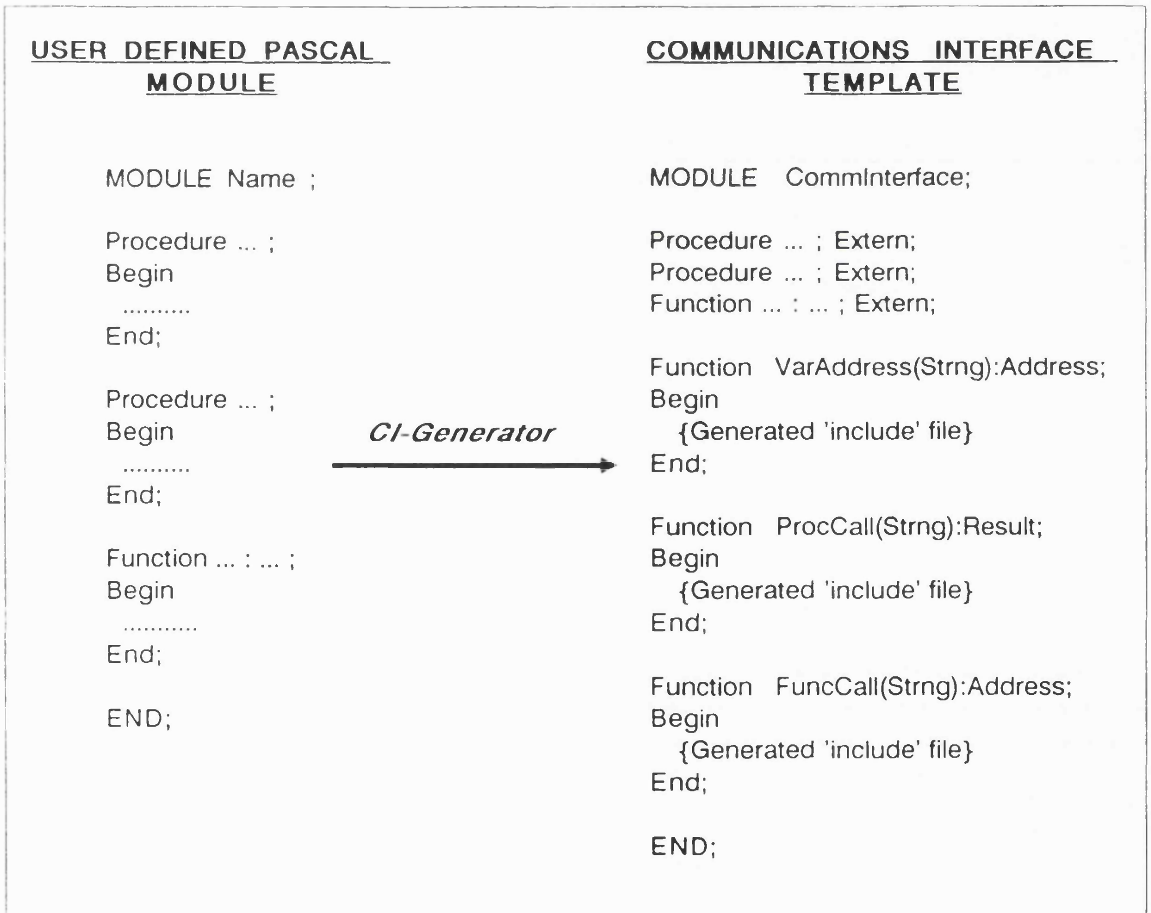
1. Variable declarations: External variables declared in the knowledge-base are extracted and added to the template file. (figure 38)



```
EXTERNAL  Time : Integer ;        VAR  Time[Extern] : Integer ;
          Dur  : Real ;                 Dur  [Extern] : Real ;
```

FIGURE 38 MAPPING BETWEEN SIMULATION AND ES VARIABLES

2. **Routine declarations:** Procedures and functions called from within the knowledge-base must be declared as 'external' to the communications interface module. The CI-generator extracts the procedure and function names by scanning the module in which they are declared. The name of the module is itself extracted from the knowledge-base by searching for the 'Pascal File' command (see section 4.4.1).



```
MODULE Name ;

Procedure ... ;
Begin
  ..........
End;

Procedure ... ;
Begin
  ..........
End;

Function ... : ... ;
Begin
  ...........
End;

END;
```

```
MODULE   CommInterface;

Procedure ... ; Extern;
Procedure ... ; Extern;
Function ... : ... ; Extern;

Function  VarAddress(Strng):Address;
Begin
   {Generated 'include' file}
End;

Function  ProcCall(Strng):Result;
Begin
   {Generated 'include' file}
End;

Function  FuncCall(Strng):Address;
Begin
   {Generated 'include' file}
End;

END;
```

FIGURE 39   THE COMMUNICATIONS INTERFACE CODE GENERATOR

3.**Code for returning variable addresses:** The CI-generator creates the code for the 'VarAddress' function using the variable names and types identified from the declaration section of the knowledge-base (figure 39). A variable name is passed to the function as a text string parameter. The function then returns the memory address at which the value of the variable is stored. When the ESSIM expert system is first activated, the 'external' variable names are in turn passed to the 'VarAddress' function and the variable addresses returned, added to the address tree (section 4.4.1). The inference engine can then gain access to variable values by searching the tree for the relevant memory address. The content of the memory address can subsequently be either read or overwritten.

4.**Code for procedure calls:** The CI-generator creates the code for the 'Proc_Call' function that enables procedures to be activated from rules defined in the knowledge-base (see figure 40). The 'Proc_Call' function receives the procedure name as a parameter and passes control to the procedure. The 'Proc_Call' function then returns a boolean value to the expert system indicating whether the procedure was found to exist. The 'Proc_Call' code is based on the procedure declarations extracted from the Pascal file declared in the knowledge-base.

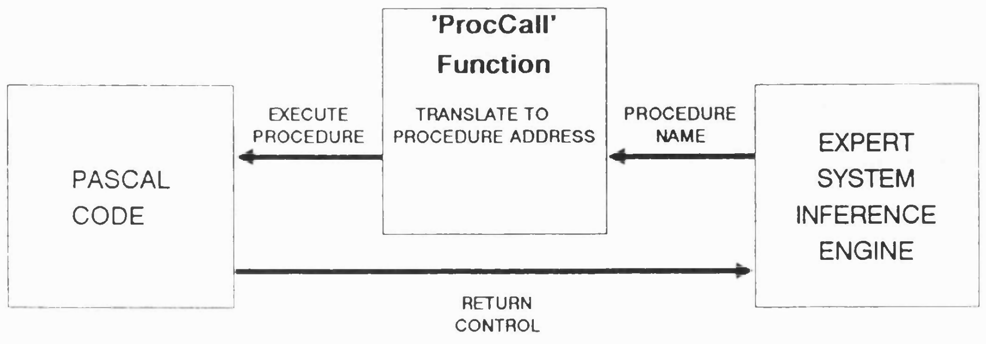


FIGURE 40 CALLING PASCAL PROCEDURES FROM ESSIM's EXPERT SYSTEM

5. **Code for function calls:** The 'Func_Address' routine receives a function name as a parameter from the expert system, calls the function, and then returns the address at which the function result is stored (see figure 41). If the parameter to the routine is invalid, a 'null' address is returned. The use of addresses permits the function result to be of any type. The value returned by the function is typically used in a calculation in the condition section of a rule. Ensuring that all variable types used are compatible is the responsibility of the modeller.

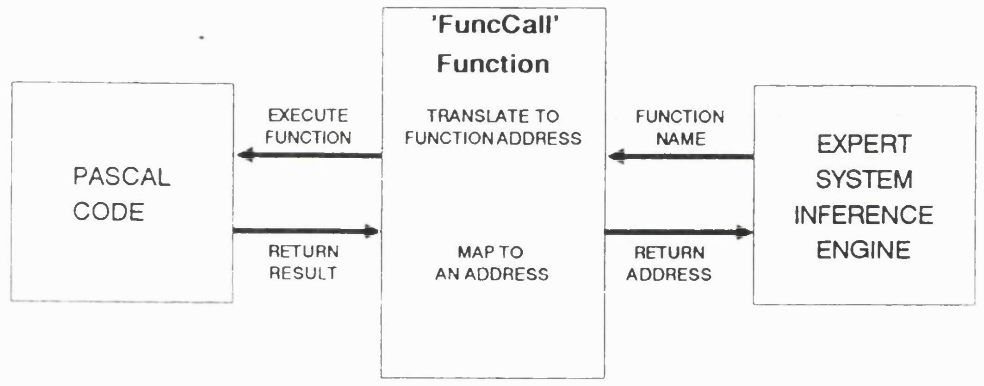


FIGURE 41 CALLING PASCAL FUNCTIONS FROM ESSIM's EXPERT SYSTEM

4.7 THE MAN-MACHINE INTERFACE

The provision of powerful graphic handling facilities on current microcomputers has lead to a significant proportion of software development resources being placed on the creation of effective man-machine interfaces. A large number of software products now make use of windows as a means of structuring menu displays or as a tool for displaying logically distinct output simultaneously on the same screen. The development of windowing environments stems from advances in parallel computing and multitasking operating systems which often require the simultaneous display of unrelated data.

The job-shop modelling environment, the predecessor to ESSIM (appendix A), made significant use of a window based display, the lay-out of which was based on the Turbo-Prolog interface. The interface was designed to enable the modeller to adapt the job-shop model to different manufacturing environments by altering the specification of the products being manufactured and other characteristics of the production plant. Windows were also used for the graphical display of simulation output and for the summarisation of model performance in terms of orders outstanding and orders completed. The job-shop modelling environment was seen to gain significantly from the provision of an effective window based man-machine interface. These gains mostly arose because of the intended role of the system as a tool for decision-support and the consequent need for a means of effectively communicating information.

ESSIM was designed as an tool for the creation of complex modelling environments such as the job-shop in which the modeller has the added benefit of being able to represent and experiment with alternative decision rules. The analyst/programmer using ESSIM consequently needs to develop a man-machine interface that permits the modeller to implement model changes and analysis model output with minimum difficulty. The development of a user friendly interface based on the use of windows is time consuming and necessitates of the programmer significant skill in low-level hardware control. The problem is compounded by the need to use Microsoft Pascal in creating the simulation code. MS-Pascal is a straightforward implementation of ANSI Pascal with no facilities provided for cursor control, colour selection, or graphics.

4.7.1 ESSIM library of low-level routines

The first step in developing the man-machine interface was to write the necessary low-level routines for the manipulation of textual output and for the creation of high-resolution graphs. These routines are available to the ESSIM user in a library and are linked to the developed code after compilation. The names used for the routines, wherever possible, are identical to Turbo-Pascal commands thus improving language compatibility and easing the translation of program code.

4.7.2 The graphics display module

ESSIM's Graphics Display Module (GDM) is accessible from the simulation model, expert system, or any other linkable program module. The GDM provides the necessary code for the development of the man-machine interface and makes use of the library of low-level video display routines. The routines can be characterised as followed:

1.Windowing routines: The GDM supports the creation of multiple overlapping windows in either text or graphics mode. A window is created by specifying screen coordinates and an associated name. The window routine then calculates the number of eight bit memory locations needed in storing the content of the display area immediately beneath the window. A request is made for the necessary amount of memory space, and the content of the screen area affected by the window, copied to the reserved RAM. When removing the window from the display, the reverse process is carried out, and the allocated memory space finally released. The GDM also provides routines for moving windows around the screen, scrolling the content of windows, and re-directing input and output operations to specified window locations.

2.Iconics: The popularity of the operating system used in the Apple Macintosh range of microcomputers has rested on the power and simplicity of it's mouse and icon driven interface. The GDM consequently provides facilities for the creation and display of user defined shapes, symbols, and character sets. Iconics can easily be combined with mouse handling routines in creating a powerful man-machine interface. For instance, a mouse routine can be used in detecting the position of the mouse 'pointer'. If the pointer is located immediately above a specified icon, the display attribute of the screen area in which the icon is displayed is reversed, and the routine associated with the icon is activated.

3.Graphics: The GDM supports graph drawing facilities which are designed to be used in combination with high-resolution window displays. The lowest level routines are designed for drawing individual pixels, lines, and circles. These drawing routines can be used in either of two modes. The first mode ensures compatibility with the underlying operating system by using the low-level BIOS routines that reside on a ROM chip. A program developed using ESSIM will consequently run on any future versions of the DOS operating system released by IBM or Microsoft. The BIOS mode also ensures ESSIM's compatibility with OS/2 and enables ESSIM to be used concurrently with other software. The second mode is designed to maximise the speed at which lines are drawn by by-passing the BIOS routines and writing directly to the video display. This prevents the use of ESSIM as a concurrent process because the screen output cannot be controlled by the underlying operating system. High-level routines are also provided for creating graph axes, drawing line graphs and bar charts, extending existing graphs, and re-scaling or shifting images. Graph lines that exceed the limits of the axes are clipped using an algorithm based on Cohen's method. (An example screen display is given on page 59)

4.7.3 The man-machine front-end module

The front-end module controls the activation of the ESSIM modules. The design and coding of the front-end is carried out by the analyst/programmer using routines from the graphics display module and ESSIM library of low-level routines. The front-end is typically window based and may provide facilities for selecting files from directories, initialising files, and setting parameters to the simulation run. A typical screen display is shown in figures 57, 58 and 59.

The creation of the man-machine front-end is nevertheless a complex process requiring repeated compilation of the module in achieving the desired screen lay-out. The process is further complicated when pull-down menus are used in combination with the mouse or when windows are designed to overlap. A program generator for creating the front-end menus and defining the location, size and content of the windows was consequently thought to be a desirable and necessary feature of ESSIM.

4.7.4 Designer

Though useful, the library routines were found to be time consuming to use and the process of designing a screen display laborious. A program generator, 'Designer', was therefore developed which permits screen designs to be created and the corresponding code generated automatically. The concept behind 'Designer' was not just to provide a conventional interface definition language, but to let the modeller create an interface interactively. 'Designer' is a form of 4GL in which 'interactive programming' is used to generate PASCAL program code. Once created, the interface lay-out can be 'edited' and new code produced. Furthermore, the eventual user of the program can be directly involved with the setting out of the interface and the presentation of the output.

The standard 'Designer' interface is based on the use of high-resolution graphics. Characters shapes are user defined and options are selected using a mouse. All input and output, whether in graphic or character format, is displayed in 'pull-down' or 'pop-up' windows. The top two lines of the screen are reserved for default menu options. The bottom line is used for the display of instructions.

The default menu options are specified by simply typing the appropriate text. Two or more spaces indicates the start of a new option. The position of the menu options is automatically adjusted such that an even lay-out is always obtained. Pointing the mouse icon at an options results in its display characteristics being reversed.

Windows can be created using the mouse. Once created, windows can be re-positioned and adjusted in size. Pull-down menus are simply created by typing text into existing windows. The pull-down menus are immediately functional permitting the screen design to be evaluated prior to generating the interface code. Pull-down windows can be also be stacked. Hence selecting an entry in one window results in another pull-down menu being displayed.

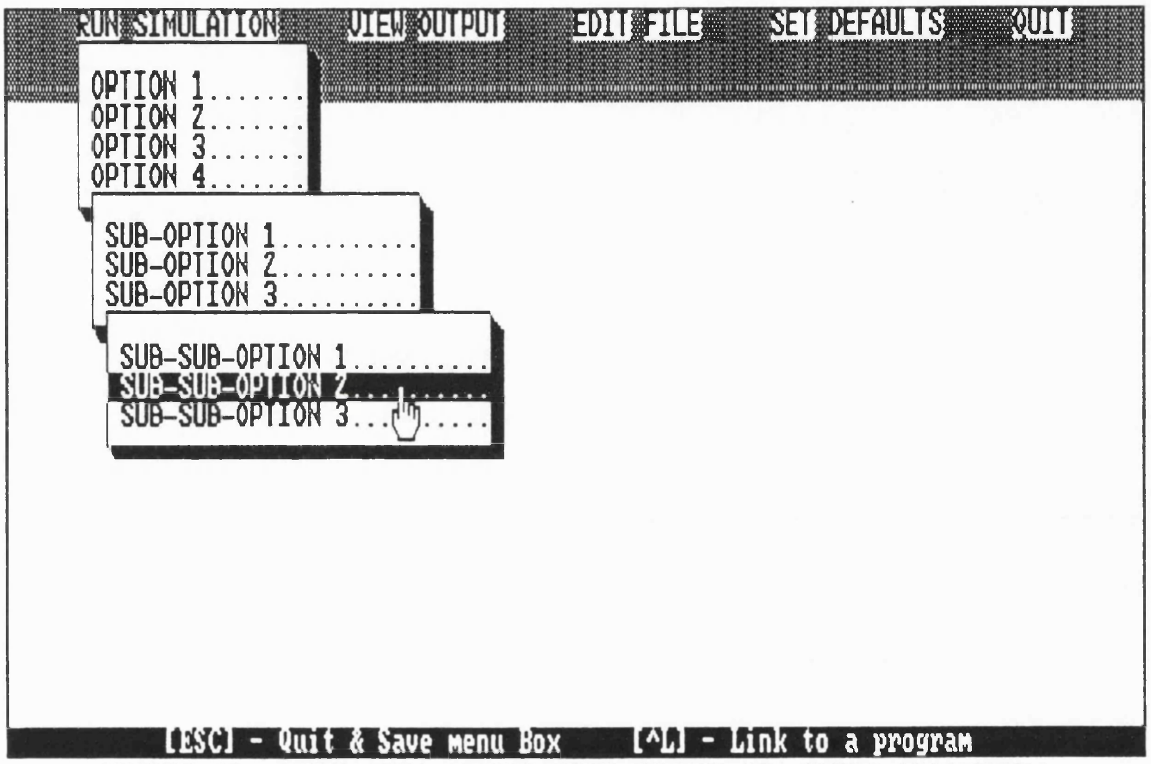


FIGURE 42 GENERATING APPLICATION INTERFACES USING 'DESIGNER'

'Designer' supports the creation of Pop-up windows. Pop-up windows are not used for the display of menus but rather for the display of free-form text, requests for user input, and the creation of graphical forms. Each window is identified by a unique code such that these can later be manipulated by the programmer(s) developing the simulation or expert system modules.

External programs can be activated by associating a file name with a menu option. Several types of program calls are possible. The modeller can 'chain' or 'spawn' a program, with, or without the use of parameters. Alternatively DOS commands can be activated, again with the possibility of parameters passing.

Output generated by external programs can either be displayed on a clear screen(in text-mode) or re-directed to a specified pop-up window. 'Designer' permits the program calls can be tested straight away without having to generate and compile the code. Designer's ability at executing DOS commands is particularly useful in the context of providing the modeller with facilities such as directory listings (possibly to a window), file copying/backup, changing default directories, and so on.

Having designed the interface, the corresponding program can be generated. The user is prompted for a file name. 'Designer' then generates the PASCAL code and compiles it to 'EXE' format. A 'screen design' file is also generated which can be used to re-load a previously designed interface. There are two ways of modifying 'Designer' files. Re-loading the screen design file or altering the Pascal code. The Pascal code can be customised by modifying the 'Designer' interface module. The module consists of a single procedure containing a CASE statement. The CASE statement entries relate to window and menu options. By inserting library commands and/or procedure and function calls, particular menu options can be made to activate given tasks.

Having designed an interface, the modeller has to place the necessary calls in the simulation code and/or expert system knowledge base to activate the appropriate windows. The read and write statements used in the model also have to be altered so as to redirect the I/O to the appropriate windows. Graphs can also be produced within windows using the appropriate library functions.

4.8 THE CODE LINKER

The use of modules complicates the process of generating the executable program. The expert system module has to be linked to the simulation code, 'Designer' interface, and appropriate libraries. To simplify the process and permit the inexperienced programmer to implement changes to the model, a separate code linking program was written.

The user is first prompted to specify the simulation model file name, the expert system knowledge base name, and the 'Designer' interface file name. The interface file name can be omitted if 'Designer' is not being used. The user has the option of specifying additional library names which may have been created for use with the model. The files can also be compiled during the execution of the program if this has not already been done. The code linker generates the appropriate commands for the Microsoft compiler, generates the executable program and then offers the modeller the option of immediately running the model.

4.9 CONCLUSION

The development of the job-shop application in co-operation with the 'Instituto Nacional de Tecnologia' provided essential practical experience in the difficulties associated with the alteration of model logic. The system also highlighted the necessity for a user friendly interface permitting the inexperienced modeller to use the model unaided and interpret output through summary reports and dynamic graphical displays. The job-shop model was implemented as an adaptable system that could be tailored for used in a range of job-shop based manufacturing concerns.

Though designed as a re-usable system, the job-shop model did not achieve the desired level of generality. To be of use in a wider context, the simulation environment had to be applicable to any real-world situation.

Such requirement spurred the development of ESSIM, in which an expert system is used for the specification of the logic applied by key individuals involved in the control of the real-world environment. ESSIM is a development environment for use by experienced modellers. ESSIM permits the creation of models similar to that of the Job-Shop, with added flexibility given to the modeller by permitting model changes through alteration of the expert system knowledge-base.

The use of a commercial off-the-shelf expert system did not satisfy the requirements of ESSIM because of the lack of adequate interfaces to 3rd generation languages and the inability to customise the structure of the knowledge-base. ESSIM's expert system module operates on the principle of a purely backward chaining inference strategy in which the goal to be resolved is defined using a Pascal function. The structure of the knowledge-base differs radically from that of existing expert systems in that 'rule-sets' are used to 'localise' knowledge. Rule-sets typically define the knowledge pertaining to a specified individual and improve the interpretability of the knowledge-base by imposing limited structuring. The use of Rule-sets also improves the performance of the expert system by limiting the search space during the inference process. ESSIM's expert system provides other features essential to the simulation task. Namely the sharing of variables with compiled Pascal code, the ability to incorporate procedural code, and the output of inference traces.

ESSIM's simulation component makes use of the discrete event three-phase approach to model development and is derived from the eLSE routines used for teaching simulation at the LSE. To warrant the use of an expert system, ESSIM models are necessarily substantial in size. The model 'template' used in conjunction with the eLSE routines was consequently modified so as to permit model development on a modular basis (see section 4.5). Additional functions were also devised permitting calls to the expert system. The development of substantive models requires the use of different model building techniques, particularly in the management of variables. The necessary disciplines in interfacing the constituent code modules were consequently identified. The changes made to the eLSE routines and the ability to construct models on a modular basis represent a significant improvement on the existing model development techniques.

The creation of an interface between compiled simulation code and an interpreted expert system knowledge-base represented a considerable challenge. The interface had to permit the sharing of common variables and the transfer of control between procedural and declarative code. The necessary generality of the interface resulted in the need for a code generator (the CI-Generator) which would scan the declarations section of the knowledge-base, identify the procedures and functions called from the expert system, and thereby construct the necessary interface module in Pascal. The interfacing of expert system and simulation model represents a significant improvement in the power and flexibility of traditional simulation modelling. The development of the interface should also be of interest to those working in AI. For instance, an expert system could be tested using realistic input data generated by a simulation model. More generally, the development of applications based on a combination of declarative and procedural code could be of benefit to a range of disciplines in which 3rd generation languages have traditionally been used.

The development of the Job-Shop application highlighted the need for an effective means of communication between application and modeller. The development of the man-machine interface was found to be a complex process requiring considerable expertise in low-level programming. State-of-the-art techniques such as windowing, icon handling, and mouse control were complex to re-produce and difficult to make use of during the development of models. One of the aims in developing the ESSIM environment was to go further than simply providing necessary libraries of routines but to actually help in the almost equally complex process of using the code. The 'Designer' interface code generator was consequently developed which allows the screen design to be produced interactively. 'Designer' reduces the time normally associated with creating complex user interfaces and permits the programmer to create a graphical front-end to a simulation model with relatively little difficulty.

The development of ESSIM would have been difficult to achieve without a practical context to which the developed theories could be applied. This practical context was first provided by the job-shop application and later by the container port model described in some detail in the following chapter.

CHAPTER FIVE

VALIDATION OF ESSIM USING A CONTAINER PORT MODEL

5.1 INTRODUCTION

Chapter four sought to describe the research steps undertaken in the development of ESSIM and provided a detailed explanation of the physical design of the eventual system. Chapter five will describe the process applied in validating the ESSIM approach to modelling. These are broadly outlined in figure 45(a).

(1) Development of a model of a computer controlled container port using conventional discrete event modelling tools.

(2) Implementation of the container port model using ESSIM with decision rules segregated within the expert system knowledge-base.

(3) Validation of the ESSIM port model output by means of comparison with the model output from the conventional model developed as part of step one.

(4) Implementation and execution of a range of experiments to be used in evaluating the ESSIM system.

(5) Development of a further version of the container port model using conventional programming techniques to replicate the functionality of the expert system.

(6) The experiments identified and carried out in step four are repeated using the different versions of the port model developed in steps one and six.

(7) Formalisation of the conclusions drawn from the research stages and experiments carried out in steps four and six.

FIGURE 45 (a) Stages in the validation of the ESSIM design

The sections of this chapter broadly follow the research and validation stages identified in diagram 45(a).

In section 5.2., the design and operational characteristics of the container port to be modelled using ESSIM is described. The container port was used in preference to the Job-Shop environment in assessing the ESSIM design as there was a lack of operational knowledge on jobbing techniques. The Job-Shop had been used in building the prototype system described in section 4.2 which used a forward chaining expert system in representing jobbing rules. Appendix A provides a fuller explanation of the functionality of the Job-Shop system.

The details relating to the design and operational characteristics of the container port were provided by a company called Highland Participants PLC. Highland Participants wished to develop a model of a computer controlled, and in parts un-manned container port which was to be constructed on the Isle of Grain. The aim of the simulation study was to simulate the expected performance of the port using alternative designs, equipment of varying specification, and alternative control procedures. Once formalised, control procedures could be extracted and later be integrated within the port's computer control system. The container port project provided a factual real-world modelling problem which could be used to realistically assess the ESSIM approach.

Conventional modelling techniques were used in the development of the first version of the container port model. Decision rules were later stripped out and embedded within ESSIM's expert system knowledge base. By using constant activity durations, the conventional model was then used as a means of validating the output from the ESSIM version of the port model. Errors in the

ESSIM program code which would otherwise have been difficult to detect were identified in this way.

The development of the second version of the port model under ESSIM is described in section 5.3. In this section, particular emphasis is placed on the describing the modular code construct which had to be adopted in implementing the container port model which is particularly substantive in size. Section 5.4 describes the knowledge-base component of the port model. Also described, are the alterations made to the expert system which resulted from difficulties in implementing certain management rules. The structure of the expert system inference engine itself and the syntax of the associated rules were explained in chapter four.

The third major component of the ESSIM port model, the graphical user interface, is described in section 5.5. The process applied in generating the interface using the 'Designer' program is explained and the resulting modeller's tools which include graphs, textual displays and pull-down menus are described. In the context of the ESSIM container port model, the graphical user interface is used in controlling the execution of the model, displaying simulation results during model execution and in allowing the modeller to edit the expert system rule-base.

Once the ESSIM version of the port model was complete, rigorous testing was required to ensure that, for instance, the expert system inference process was being executed correctly. Model validation was achieved by comparing the output from the ESSIM model with that of the original port model which had been entirely coded in Pascal. Constant activity durations were used to ensure that at every time step, the behaviour of both models could be expected to be identical. The validation of the ESSIM port model is explained in section 5.6.

Having completed and validated the ESSIM version of the container port model, work commenced on formally evaluating the cooperative simulation and expert system approach to modelling as encompassed in the ESSIM design. The first step undertaken was to devise a range of experiments which could be used in assessing the impact of introducing changes to the expert system knowledge base and simulation modules. These experiments can broadly be classed as follows:

(1) Experiments involving changes to knowledge-base rule parameters.
(2) Experiments based on changes to variable values within knowledge-base rules.
(3) Introducing model changes which require the re-organisation of existing decision rules or the introduction of new decision rules.
(4) Experiments requiring major model changes with specific reference to the introduction of new entity cycles.

Section 5.7 is sub-divided to cover each of these class of experiments. Each individual experiment is explained in detail and initial findings are reported relating to the ease or difficulty with which model changes were implemented.

Defining and carrying out experiments using the ESSIM version of the container port model is insufficient for the purpose of fully assessing the value of the ESSIM approach to modelling. Further evidence was therefore sought by comparing the results of the experiments with those obtained by repeating the experiments using alternative modelling techniques. A further version of the container port model was therefore developed in which the rules associated with each port manager were hard-coded within individual Pascal functions. These pascal functions were isolated from the main body of the simulation program in order to replicate the concept of a knowledge-base representing decision makers, independent from the simulation component depicting physical entities, activities and their inter-relationships. This new version of the container port model is fully described in section 5.8.

The experiments described in section 5.7 which had been applied to the ESSIM port model were repeated using the new version of the container port models. The experiments were also applied to the first port model which had been developed entirely using conventional three-phase discrete event routines. The conclusions from these further experiments are reported in section 5.8.1. The conclusions are based on a comparison of the accuracy, adaptability and maintainability of the model representations.

From the comparison of the characteristics of each individual version of the port model, conclusions are drawn as to the merits and limitations of each of ESSIM's components. The Pascal simulation routines themselves are first examined. Some unusual concepts had been introduced such as the use of individual modules in representing each entity cycle. The value and limitation of these new modelling concepts are discussed and reported in section 5.9.1. A similar approach is taken in evaluating the benefits and limitations of the expert system. The development of the expert system knowledge-base and inference engine were based on existing theories in terms of the core principles such as backward chaining. With respect to functionality, a number of fairly radical design features were implemented, particularly in terms of the interface to the Pascal language and the use of rule-sets. The benefits and limitations of the expert system are discussed in section 5.9.2. The impact of the user-interface and the benefit conveyed by the 'Designer' code generator are similarly debated in section 5.9.3.

The conclusions to this chapter are presented in section 5.10.

5.2 DESIGN OF THE CONTAINER PORT

Several alternative port designs were considered by Highland Participants for the Isle of Grain site. One of these designs was eventually used in creating the ESSIM model and will now be described.

The following numerical data are averages used in the model. These constants were used to simplify the validation of the model output.

A ship reaches each of the port's berths every 24 hours. The impending arrival of a ship is notified to the port authorities 4 hours in advance. Each ship takes 2 hours to dock and two hours to leave the port. There are two berths.

Cranes are used to unload containers from the ships onto waiting Internal Movement Vehicles (IMVs). One of the ship berths is served by two cranes, and the other by three cranes. A crane takes 40 seconds to load a container from a ship onto an IMV or from an IMV onto a ship.

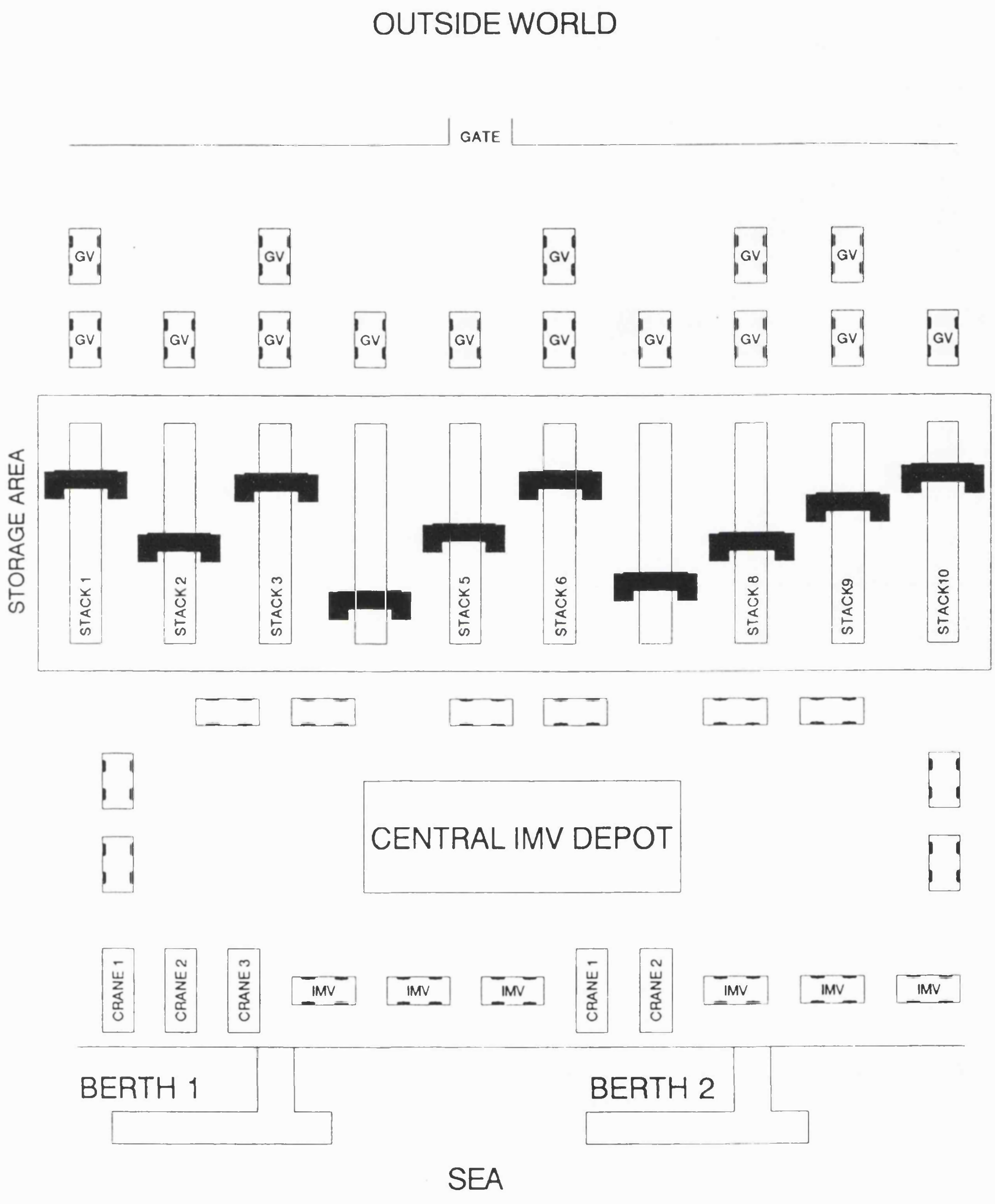


FIGURE 43 LAYOUT OF THE CONTAINER PORT

Each ship is expected to carry the maximum load of 500 containers. 500

import containers are unloaded from each ship, these being replaced by 500 export containers. In some cases, ships may arrive empty to collect a consignment. On arrival of a ship, all cranes are set to 'import', and the process of unloading containers begins. Once all import containers have been unloaded, the cranes switch to 'export' mode and the loading of export containers onto the berthed ship starts. In the case of the berth which has three ship cranes, only two of the cranes can work at any one time in 'export' mode. There are other loading/unloading scenarios which provide management with options in the optimisation of the ports operations. The scenario which is investigated in this thesis is one in which import and export operations occur in parallel. As space is cleared in the ship's hold, it becomes possible to load export containers before all import containers have been unloaded. This is represented in the expert system knowledge-base by rules which set the number of ship cranes operating on imports/exports to being proportional to the remaining workload. Hence, in the case of two ship cranes, the start condition is that both cranes work on imports only. If the number of remaining import containers falls below half the total import workload, one of the cranes is shifted over to working on exports. When no import containers remain, both cranes work uniquely on exports.

There are 100 IMVs that can transport import containers from a berth to the storage area, and export containers from the storage area to a berth. IMVs that are not currently in the system wait in a central depot. Once an IMV leaves the depot, it becomes allocated to servicing one of the berths. An IMV that has just delivered a container to a ship will either return empty to the storage area or will wait its turn to collect an import container. If the cranes are all working on exports, the IMV returns empty to the stack. In some circumstances the relative number of IMVs servicing the berths may vary. For instance, if no empty IMVs are available at one of the berths, an idle IMV may be transferred from the other berth. If neither berth has any empty IMVs, an allocation is made from the central depot. In both cases, the transfer of an IMV takes 40 seconds. IMVs return to the central depot when both berths are unoccupied.

A loaded IMV takes 60 seconds to travel from a berth to the storage area. When empty, the same journey takes 30 seconds. An IMV takes 60 seconds to travel empty from the storage area to a berth. When loaded, the IMV takes 120 seconds.

The storage area is divided into 10 sub-areas, each referred to as a stack. An empty IMV returning to the storage area must collect containers for the ship to which it has been allocated. Export containers are typically scattered across all of the storage area. Consequently the IMV moves to the stack at which the most export containers remain. If the ship is fully loaded or no export containers can be found in the storage area, the IMV returns to the central depot. Full IMVs moving towards the storage area are allocated uniformly between the stacks.

An IMV which arrives empty at a stack may be re-allocated to service another berth. This could happen in a situation where the export workload for the ship currently being serviced is lower than the export workload of the ship in the other berth. In such a situation, the empty IMV may be transferred to another stack in order to maintain a balance in the number of export containers per stack. Occasionally, empty IMVs waiting in the storage area may return to a berth without collecting an export container. This may happen if the total number of ship cranes working on imports exceeds the number of ship cranes working on exports. Furthermore, the IMV may return to a different ship if there happens to be an imbalance in the import workload at the two berths.

A loaded IMV which arrives at a stack, waits in a queue to unload. Once free, the IMV will either return to the berth to which it has previously been allocated, or wait its turn at the stack to collect an export container. The latter tends to be the case, if the number of cranes working on exports exceeds the total number of cranes working on imports.

Gate Vehicles (GVs) are trucks which enter the port to either deposit export containers or collect import containers. GVs move from the entrance to the port to either one of ten bays associated with each of the stacks. GVs begin arriving with export containers 4 hours prior to the arrival of a ship. (Simultaneous to the arrival advance warning given by the ship). The arrival process continues at approximately 12 minute intervals until the allocation of 500 export containers per ship is received. Only once all export containers have been received do the empty GVs start arriving. Each arriving GV is associated with a specific ship and is allocated to either an import or export task.

GVs carrying export containers travel to bays allocated randomly. An empty GV moves to the bay of a stack where a container is known to be located. When there are several bays to chose from, a selection is made which ensures that the distribution of import containers remains balanced. A GV takes one minute to move to a bay.

Each stack is serviced by a single Rail Mounted Gantry crane (RMG). The RMG unloads import containers from IMVs and transfers these to free positions in the stack area. At some later time, the RMG collects the same container and deposits it on a waiting gate vehicle. The GV then leaves the port. Exports are handled in a similar way. The containers are removed from incoming GVs and transferred to free positions in the stack. When the ship is ready to accept export containers, the RMG collects the appropriate containers and deposits these one at a time on waiting IMVs.

IMVs and GVs compete for the allocation of RMGs. If import and export jobs are both pending, then priority is given to the IMV. When idle, an RMG waits in a position mid-way along the rail track. If a loaded IMV arrives at the stack, the RMG moves from the idle position to the 'shore side' of the stack in 40 seconds. The container is then loaded onto the RMG in 60 seconds. The RMG subsequently deposits the container in an available position in the stack, the average time taken being 60 seconds (this includes the time taken to return to the idle position). If an empty IMV arrives at the stack, the RMG collects the allocated export container and moves to the 'shore side' of the stack in a mean time of 60 seconds. The RMG then off-loads the container onto the waiting IMV in 60 seconds and subsequently returns to the idle position in 40 seconds. The converse situation is almost identical. If a loaded GV arrives at a bay, the RMG moves from the idle position in a mean time of 40 seconds. The container is then loaded onto the RMG in 60 seconds. The RMG subsequently deposits the container in an available position in the stack, the average time taken being 60 seconds. If an empty GV arrives at the stack, the RMG collects the allocated export container and moves to the 'shore side' of the stack in a mean time of 60 seconds. The RMG then off-loads the container onto the waiting GV in 60 seconds and subsequently returns to the idle position in 40 seconds.

5.3 STRUCTURE OF THE SIMULATION MODEL

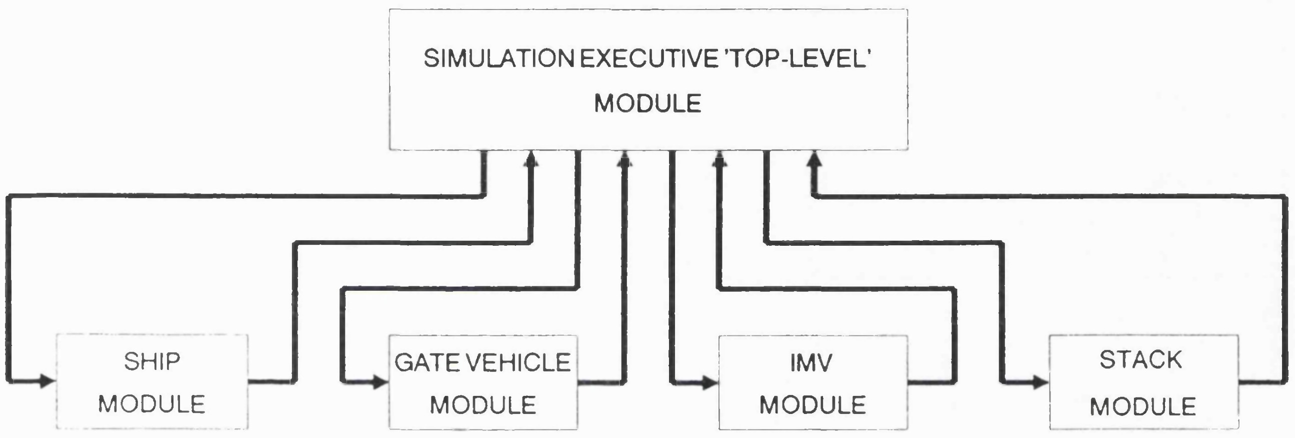


FIGURE 44 REPRESENTATION OF MODEL CYCLES USING MODULES

The ESSIM model of the port is structured on a modular basis which eases the process of writing the simulation by enabling the logical separation of code segments. Each module can be compiled separately and tested using data embedded in a 'dummy' module. Furthermore, in a PC environment, the use of a modular construct permits programs to be created which exceed the 64K limit imposed by the MS-DOS operating system.

The characteristics of the port in terms of sequences of events and estimated durations, were identified through repeated interviews with senior management. Some of the collected information was subsequently formalised using an Activity Cycle Diagram (ACD) (Hills[1971] and Clementson[1982]). The components of the ACD, in terms of interrelated life cycles, represented an ideal structure for the modularisation of the model. Consequently the ESSIM port model consists of modules corresponding to the life cycle of the main entity types. Namely, the ship, Internal Movement Vehicle(IMV), Rail Mounted Gantry(RMG), and Gate Vehicle(GV) cycles.

```
PROCEDURE C_Phase ;
 Begin
       Ship_Module_Cs ;
        Gate_Module_Cs ;
        IMV_Module_Cs ;
        Stack_Module_Cs ;
  End;


PROCEDURE Initialise_Model ;
 Begin
       Init_Ship_Module ;
        Init_Gate_Module ;
       Init_IMV_Module ;
       Init_Stack_Module ;
  End;
```

FIGURE 45 SIMULATION EXECUTIVE MODULE CALLS

The port model is controlled through a 'top-level' module which initialises system variables and subsequently manages the calls to C-events. Unless major changes are made to the logic of the model through the addition or removal of entity life cycles, there should be no need to alter the code contained in the top-level module. The content of the main routines are shown in figure 45. Calls to the C-Phase procedures are managed by the simulation executive contained in a separate library of code.

The ship cycle being one of the simplest modules will now be discussed in greater detail. The general structure of the module is defined in figure 50. The use of the eLSE routines for discrete event three-phase modelling are well documented in Chew [1986]. These will consequently not be described.

Each module has a declaration of constants which define the upper bounds of arrays used for the storage of simulation data (figure 46). These constants typically refer to the maximum permissible number of various entity types. In practice, the actual entity count is defined by the modeller in the expert system knowledge base (see section 5.4). The use of dynamic data structures such as linked lists would have eliminated the need for the definition of upper limits and would in many cases have reduces the amount of memory used. Conversely, the use of array structures simplifies retrieval of data, and improves model performance.

```
CONST
    Max_Berths = 4 ;
    Max_Ships  = 9 ;
```

FIGURE 46 CONSTANT DECLARATIONS

```
TYPE
    Ship_Details = ARRAY[1..Max_Ships] OF RECORD ;
                                          ImportLoad : Integer ;
                                          ExportLoad : Integer ;
                                          Berthed_At  : Entity ;
                                         End;

VAR [EXTERN]
    ...........
    Time : Integer4 ;
    ...........

VAR [PUBLIC]
     Address_For_Ship_Details : ADS OF Ship_Details ;
     Ship_Cycle_Interface : RECORD
                              Q_Berths_Occupied : Queue ;
                              Q_Berths_Completed: Queue ;
                             End;

VAR {Local variables}
      Q_Ship_At_Sea, Q_Ship_At_Berth, Q_Free_Berth,.... : Queue ;
```

FIGURE 47 MODULE DECLARATIONS

The introduction of a modular structure to the simulation model resulted in a need to alter the way in which the programs would normally be written. The individual modules that make-up the simulation model are physically independent from each other. In other words, simulation data is local to a module unless steps are taken to make certain data values shareable. This imposes on the modeller a far greater level of discipline in structuring code and data than would normally be required in a conventional discrete event simulation environment. For instance, in figure 47, arrays in the ship cycle module which need to be accessed by other modules need to be referenced using memory addresses. In figure 47, the variable 'Address_For_Ship_Details' is an address pointing to the start of an array defined as the 'Ship_Details' data type. This address can then be used from within any of the simulation modules to access the content of the array.

The external variable command in figure 47 is used in specifying the name of variables accessed within the module but declared in another module. Finally, queues are defined as data structures which are local to individual modules. This helps to minimise the amount of memory space used and prevents accidental changes to queue structures from within other modules. The only exception to this rule are queues which are defined within an interface. For instance, in figure 47, two queues are defined within a record referenced by the 'Ship_Cycle_Interface' variable. These queues are required by the IMV cycle module. The use of an interface in defining the link between major activity cycles clarifies the relationship between modules and serves to highlight the queues which are modifiable from within multiple code segments.

At the start of the simulation run, data structures relating to the ship cycle are initialised through a call to the 'Init_Ship_Module' procedure (figure 48). THEREARE and MAKEQ are standard eLSE routines whereas FILLQUEUE is specific to ESSIM. FILLQUEUE is used to add entities to a specified queue. Interface definitions are also initialised at this stage. In the example of figure 48, the queues defined in the interface record are set equal to two equivalent queues which are local to the ship module. The 'Queue' type is in fact the start address of a linked list. Consequently, changes made to one of the 'queues' is automatically reflected in the other 'queue'.

```
PROCEDURE Init_Ship_Module ;
Begin
   ........
   Initialise_Ships ;            {Initialisation of arrays}
   Initialise_Berths ;           {Initialisation of arrays}
   Thereare(N_Berths, Berths, 'Berths') ;
   MakeQ(QFreeBerth, 'QFreeBerth', Berth) ;
   MakeQ(QBusyBerth, 'QBusyBerth', Berth) ;
   FillQueue(Berth, N_Berths, QFreeBerth) ;
   WITH Ship_Cycle_Interface DO
   Begin
      Q_Berths_Occupied := QBusyBerth ;
      Q_Berths_Completed := QBerthDone ;
   End;
End;
```

FIGURE 48 MODULE INITIALISATION

The syntax of B-Events and C-Events that comprise the body of each module are intrinsically the same as those used in developing conventional three-phase models using the eLSE routines. The principal differences are the use of addresses in initiating B-Events using the 'Schedule' command, (see section 4.5) and the use of calls to the expert system. Figure 49 consists of a listing of the code that comprises a typical C-event in the ship cycle module. The body of the procedure consists of a single 'Schedule' instruction which is executed as long as the WHILE condition is satisfied. In a conventional eLSE model, the conditional statement would have consisted of a combination of expressions which would be used in determining whether the event should occur. In the case of the ESSIM example, 'StartShipArrive' is a function which returns a boolean value. The GOAL command within the function is used in transferring control to the expert system (see section 4.4.3). The parameters to the function identify the goal to be resolved as 'StartshipArrive' and the individual responsible for the decision as being the 'ShipManager'. The result returned is located at a memory address identified by the variable 'Res'. The use of an address for the function enables the expert system to return any standard Pascal or user defined data type as the goal result.

```
PROCEDURE  START_Ship_Arrive_At_Sea  ;

    FUNCTION StartShipArrive : Boolean ;
    VAR Res: ADS OF Boolean  ;
    VAR[Public]  NumberOfShipsAtSea : Integer ;
                 ShipArrivalDue : Boolean ;
    Begin
        NumberOfShipsAtSea  :=  QSIZE(QatSea)  ;
        IF QSIZE(QSeaOpen) > 0 THEN ShipArrivalDue := True ;
           ELSE ShipArrivalDue := False ;
        Res  :=  GOAL('ShipManager', StartShipArrive)  ;
        StartShipArrive  :=  Res^  ;
    End;

Begin
     WHILE  StartShipArrive  DO
        SCHEDULE(ADS END_Ship_Arrive_at_Sea, BEHEAD(QatSea), _Time);
End;
```

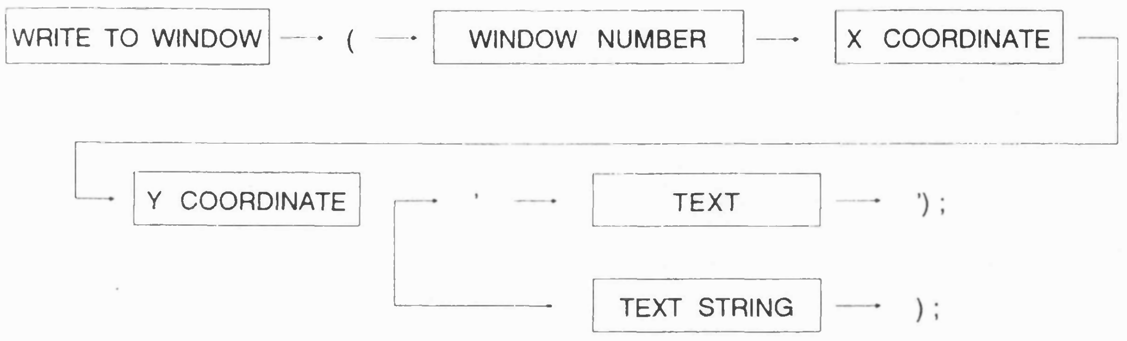
FIGURE 49 EXAMPLE C-EVENT

Although the expert system knowledge-base is interpreted and the simulation module is compiled, variable values can still be shared as if the complete model was coded in one language. Consequently, the two variables which are defined as "Public" in figure 49 can be accessed from the expert system knowledge-base as if these had been locally defined.

```
PROCEDURE START_Ship_Arrive_At_Sea ;                    {C-Event}
          Check with the expert system if a ship arrival is due.
          Add to queue 'Advance_Warning'.

PROCEDURE START_Move_To_Port ;                          {C-Event}
          Give advance warning of arrival of ship.
          Move towards the port.

PROCEDURE START_Dock_At_Berth ;                         {C-Event}
          Check with expert system if a berth is free.
          Start docking process by allocating the ship to a berth.

PROCEDURE START_Leave_Berth ;                           {C-Event}
          Check with expert system that work has been completed.
          Ship starts to leave for the open sea.

PROCEDURE END_Ship_Arrive_At_Sea ;                      {B-Event}
PROCEDURE END_Move_To_Port ;                            {B-Event}
PROCEDURE END_Dock_At_Berth_For_Ship ;                  {B-Event}
PROCEDURE END_Dock_At_Berth_For_Berth ;                 {B-Event}
PROCEDURE END_Leave_Berth_For_Ship ;                    {B-Event}
PROCEDURE END_Leave_Berth_For_Berth ;                   {B-Event}

PROCEDURE Display_Ship_Module_Output                    {Update screen display}
PROCEDURE Ship_Module_Cs                                {C-Event calls}
```

FIGURE 50 C & B EVENTS IN THE SHIP CYCLE MODULE

Figure 50 lists the procedures that constitute the ship cycle module. Procedure names beginning with the 'Start' key-word are C-Events that represent the start of an activity. Procedure names which begin with the word 'End' are time dependent B-Events which correspond to the end of activities.



```
EXAMPLES:     WRITE_WINDOW_POS( 3, 10, 4, 'IMPORTS') ;
              WRITE_WINDOW_POS( 3, 18, 4, ImportNumber) ;
```

FIGURE 51 PROCEDURE FOR OUTPUT DISPLAY TO WINDOWS

Each of the entity life cycle modules also has a procedure responsible for the display of output (e.g. Display_Ship_Module_Output in figure 50). Each of these procedures is executed once during the C phase. The screen display is updated using commands from the Designer library which permit the redirection of text to windows and the creation of graphical output. A typical command is shown in figure 51.

```
PROCEDURE Ship_Module_Cs ;
Begin
    Write_Window(2, 'SHIP_MODULE_Cs') ;
    START_Ship_arrive_at_Sea ;
    START_Move_To_Port ;
    START_Dock_At_Berth ;
    START_Leave_Berth ;
    IF ShipTrace THEN Display_Ship_Module_Output ;
End;
```

FIGURE 52 MODULE C-EVENT CALLING PROCEDURE

Each entity life cycle module has a procedure which calls the C-Event routines in sequence. (e.g. Ship_Module_Cs in figure 52). The content of a C-Phase procedure of the ship life cycle module is shown in figure 49.

5.4 STRUCTURE OF THE EXPERT SYSTEM KNOWLEDGE BASE

1. ADDITION OF A 'GOAL' COMMAND AND OTHER RELATED CODE TO THE SIMULATION MODEL.
2. DEFINITION OF 'PUBLIC' VARIABLES IN THE SIMULATION CODE.
3. DEFINITION OF 'EXTERNAL' VARIABLES IN THE KNOWLEDGE BASE.
4. DEFINITION OF 'LOCAL' VARIABLES IN THE KNOWLEDGE BASE.
5. DEFINITION OF PROCEDURES AND FUNCTIONS CALLED FROM WITHIN THE KNOWLEDGE BASE.
6. DEFINITION OF KNOWLEDGE BASE RULES.
7. ALLOCATION OF RULES TO APPROPRIATE 'RULE SETS'.
8. TESTING OF RULE LOGIC.

FIGURE 53 DEFINITION OF NEW GOALS.

As described in section 4.4.1 of chapter four, the expert system knowledge-base is divided into "rule-sets". In the context of the container port model, these rule-sets are used to group together management rules relating to a particular activity. The end result is a number of rule-sets which conceptually mirror the modular structure of the simulation model. There are consequently rule-sets for the 'Crane manager', 'IMV manager', etc..

The expert system knowledge base was developed on an incremental basis. Sets of rules were created and the effect of these tested against the original model. The additional of further sets of rules required the actions identified in figure 53 to be carried out. These steps will now be described in greater detail.

```
Write_Window(2, 'GOAL CraneJobs') ;
ResultAddr := GOAL('CraneManager', 'CraneJobs') ;
CraneWork := ResultAddr^ ;
```

FIGURE 54 GOAL CALL TO EXPERT SYSTEM

The syntax of the GOAL command which is used to transfer control from the simulation module to the expert system was described in section 4.3. An example in the context of the port model is given in figure 54. Let us now consider the GOAL instruction in figure 54 which forms part of the IMV cycle module, and its corresponding expert system rules. The expert system goal, 'CraneJobs', identifies whether there is any outstanding work for a specific crane. If the crane is idle, the expert system selects the next job to be carried out and returns to the simulation model the necessary instruction and any relevant data such as the duration of the activity. The expert system also reports whether the crane should be re-allocated from working on import containers to loading the ship with export containers.

- There is no work to be carried out by the crane when the crane is non-operational.
- There is work to be carried out by the crane when there is an IMV which can be loaded or unloaded or a ship which can be loaded or unloaded and when the crane is operational and in the correct mode.
- The crane is in the correct mode when there is an IMV which can be loaded or a ship which can be unloaded and the crane is currently allocated to working on import containers.
- The crane is in the correct mode when there is an IMV which can be unloaded or a ship which can be loaded and the crane is currently allocated to working on export containers.
- There is an IMV which can be loaded by the crane when the crane has finished picking-up an import containers and there is an empty IMV waiting.
- There is an IMV which can be unloaded by the crane when the crane is not carrying a load and is waiting idle and there is a loaded IMV waiting.
- There is a ship which can be loaded by the crane when the crane has finished picking-up an export container and there is a ship waiting.
- There is a ship which can be unloaded by the crane when the crane is not carrying a load and is waiting idle, but only on the condition that loading export containers onto the ship is not a more urgent task.
- The crane should give priority to loading export containers onto the ship when there are no further containers to unload from the ship or when there were no containers to unload in the first place. In the case of ship berth No.2, one crane must always remain allocated to working on import containers.
- Authorisation should always be sought prior to changing a crane's mode of operation from 'Imports' to 'Exports'.
- There is always a five minute delay in obtaining authorisation unless the current time is between 1pm and 2pm in which case there is a 60 minute delay in gaining authorisation.

FIGURE 55 (a) Sample operational rules.

operational rules in figure 55(a). The rules are generic to all ship berth cranes.

The rules in figure 55(a) are insufficiently detailed for the purpose of resolving the top level goal which is whether or not a given crane can commence work on a particular activity. The rules listed are only those associated with the 'Crane manager'. However, the activation of the crane requires a degree of interaction with the IMV and SHIP cycles. Consequently, in practice, some of the rules contained within the 'Crane Manager' rule-set are linked to other rules in the 'IMV Manager' and 'Ship Manager' rule-sets. For instance, the process of loading an IMV with a container from a crane may require a level of decision making by the 'IMV manager'. In certain situations, a decision may be taken by the IMV manager to transfer an IMV from another queue or to retrieve an IMV from a depot of idle IMVs. The full ESSIM knowledge-base is listed in appendix B.

In some cases it may be best to place certain rules within the Pascal code rather than the expert system knowledge-base. Certain rules which are associated with physical constraints are one such example. With respect to the rules listed in figure 55(a), it is best to check whether a crane is idle within the Pascal code rather than determining this within the expert system. For instance, a crane must always be idle before it can be allocated a new task and so this basic fact may as well be hard coded as a condition to the execution of the 'C' event in the Pascal model. Embodying this simple rule as part of the 'C' event procedure eliminates the need to call the expert system when the crane is busy, thereby improving the performance of the model.

```
RULESET CraneManager (INHERIT ImvManager, ShipManager) ;

[*] NumberOfShipCranes = 5 ; {Total number of ship cranes}

[*] TimeToLoadShip = 40 ;

[*] TimeToUnloadShip = TimeToUnloadCalc ; {Call to Pascal functiion}

[1] CRANEJOBS = False IF CraneOperational = False ;

[2] CRANEJOBS = True WHEN ((_LoadImv = True) OR (_UnloadImv = True)
                          OR (_LoadShip = True) or (_UnloadShip = True))
                          AND (CraneInCorrectMode = True) ;

[3] CraneInCorrectMode = True WHEN (CraneOperational = True)
                        AND ((CraneOnImports = True)
                          AND ((_LoadImv = True) OR (_UnloadShip = True)))
                         OR ((CraneOnImports = False)
                          AND ((_UnloadImv = True) OR (_LoadShip = True)));

[4] _LoadImv = True WHEN (CraneLoaded = True) AND (EmptyImvToLoad = True) ;

[5] _UnloadImv = True WHEN (CraneLoaded = False) AND (_FullImvToUnload = True) ;

[6] _LoadShip = True WHEN (CraneLoaded = True) AND (ShiptoLoad = true) ;

[7] _UnloadShip = True WHEN (_ChangeACraneToExports = False)
                           AND (CraneLoaded = False) AND (ShipToUnload = True) ;

[8] (_ChangeACraneToExports = True) AND (GetAuthorisation = True)
                            AND (TimeToGetAuthorisation = CalcFromCurrentTime) {Pascal}
                         AND (CraneOperational = False)
                          ~ (_ChangeACraneToExports = False)
                         AND (GetAuthorisation = False)
                          IF (CurrentShipberth = 1) AND (NumCranesOnImports > 0))
                           OR ((CurrentShipberth = 2) AND (NumCranesOnImports > 1))
                          AND (NumImportsRemaining = 0) AND (NumTotalImportJobs > 0);

[9] (AUTHORISECRANETOEXPORT = True) AND (CraneOperational = True)
                         AND (CraneOnImports = False)
                          ~ (AuthoriseCraneToExport = False)
                          IF ((CurrentShipBerth = 1) AND (NumCranesOnImports > 0))
                           OR ((CurrentShipBerth = 2) AND (NumCranesOnImports > 1))
                          AND (NumImportsRemaining = 0) AND (NumTotalImportJobs > 0);
```

FIGURE 55 THE 'CraneManager' RULESET

Figure 55 is a listing of the expert system rules which are used in determining the next job to which a Berth crane is to be allocated. The rules are equivalent to those listed in figure 55(a). The goal to be resolved is defined within the 'CraneManager' Ruleset as the boolean variable 'CraneJobs'. The first three statements in the CraneManager Rule-set, which have no associated rule number, could have been defined within the Pascal code. The Advantage in declaring certain variables within the knowledge-base is that the values can be changed during repeated executions of the model without having to edit and re-compile the Pascal simulation code. The same benefits can be achieved using a Pascal data file, though in practice maintaining the values within the knowledge-base is somewhat neater. (e.g. No additional coding is required in declaring values in the knowledge-base and the expert system also validates the syntax of any entries). The second statement in figure 55 defines the time taken by a crane to load a container onto a berthed ship in seconds. The time value can be changed manually prior to each new execution of the model or the modeller can define rules which will have the effect of modifying the time value based on the outcome of specific decisions.

The first step in defining the rules associated with the goal 'cranejobs' in figure 55, was to identify the variables that the expert system would require in carrying out the inference. These variables, if not already accessible across multiple code modules had to be defined as shared variable using the Microsoft Pascal 'Public' identifier. Correspondingly, the same variables had to be defined as external to the expert system using ESSIM's 'External' variable identifier (see figure 38 in chapter 4). All other variables required during the inference process were declared as 'Local' to the expert system. Algorithms that could not be defined using ESSIM's restricted syntax were written using Pascal procedures and/or algorithms and declared to the expert system using the 'Pascal rule' command (see figure 56). Other rules are necessary in resolving the goal but are defined in the ImvManager and ShipManager rule-sets.

```
EXTERNAL
    NumberOfShipCranes, TimeToLoadShip, TimeToUnloadShip, CurretnShipBerth,
    NumCranesOnImports,.......... : INTEGER ;
    CraneOperational, ImvsIdle, _LoadShip, _UnloadShip, _LoadImv, _UnloadImv,
    CraneOnImports,.......... : BOOLEAN ;

LOCAL
    CraneJobs,........... : BOOLEAN ;

PASCAL FILE 'Rules.pas' ;
```

FIGURE 56 EXPERT SYSTEM DECLARATIONS

In figure 55, rule 1 specifies that there are no 'CraneJobs' if the crane is non-operational. The IF rather than the THEN condition is used. This is because one could not deduce from the fact that a crane was operational that there was consequently work for the crane to do. The inference engine attempts to execute rule 2 if the goal cannot be resolved. Rule 2 specified that there is work for the crane if an IMV or ship can be loaded or unloaded. A crane is either allocated to import containers or export containers and so it is also necessary to determine whether the crane is in the correct mode of operation to permit it to carry out the next specified job. Rule 3 verifies if the crane is in the correct mode of operation. The use of a WHEN type rule results in the

'CraneJobs' goal returning the value FALSE if the conditional statement cannot be satisfied. The variables used in the conditional statement are declared as 'External' permitting the simulation model to determine their value. The use of an underscore as the first character of the variable names indicates that the variables should initially be set to have an undefined value. Consequently, in attempting to satisfy rule 2, the inference engine in turn sets '_LoadImv', '_UnloadImv', '_LoadShip', '_UnloadShip' and 'CraneInCorrectMode' as sub-goals.

Rule 4 attempts to determine whether an IMV can or cannot be loaded. The crane has to be working on imports and loaded with a container. Naturally, an empty IMV must also be available. The 'EmptyImvtoLoad' variable is defined as being local to the expert system. The inference engine cannot resolve the rule until a value has been associated with the variable. Consequently, the inference engine sets 'EmptyImvToLoad' as the third level sub-goal. IMVs are the responsibility of the IMV manager and not the crane manager. Consequently, the rules necessary in resolving the sub-goal are located in another rule-set. To summarise these, an empty IMV can be loaded if an empty IMV is idle in the queue at the berth, if an empty IMV can be transferred from the other ship berth, or if an idle IMV is available in the central depot.

Having resolved the third level sub-goal 'EmptyImvToLoad', the inference engine returns to the second level sub-goal which was '_LoadImv'. This sub-goal can now also be resolved. Nevertheless the top level goal still cannot be satisfied as a value is required for the '_UnloadImv' boolean variable. The inference engine identifies rule 5 as a potential means of satisfying the new sub-goal. Once again, the rules concerned are the responsibility of the IMV manager and are located in another rule-set.

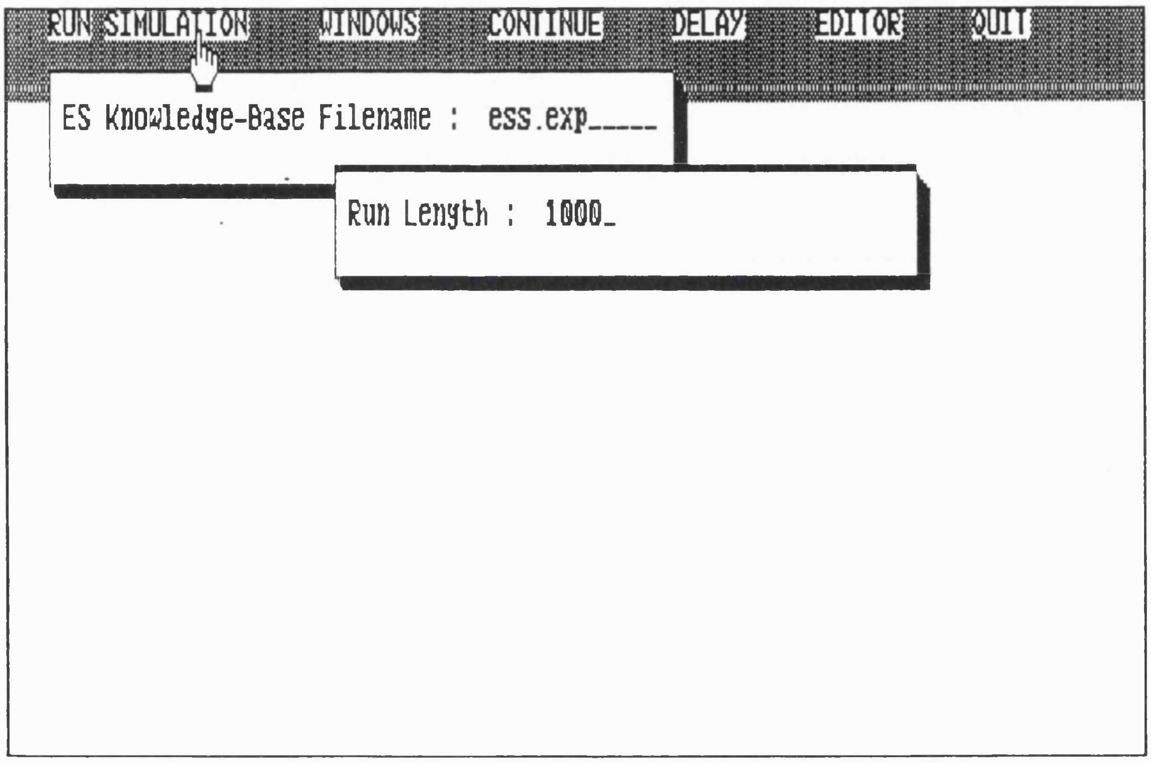
The remaining sub-goals of rule 2, namely '_LoadShip', '_UnloadShip' and 'CraneInCorrectMode' are resolved by rules 6, 7 and 3 respectively. Rule 6 specifies that a ship can be loaded if the crane has already lifted a container from an IMV and the ship is waiting in the berth. The duration for the loading process is set in the second statement of the ruleset by associating a value with a public variable. Rule 7 specifies that a ship can be unloaded if the crane is waiting idle and is not carrying a load, the ship is at the berth and the crane is not about to be re-allocated to exports. The last of these conditions is resolved by rules 8. Rule 8 specifies that a crane working on imports should change to working on exports once all import containers have been unloaded from the holds of the ship. In the case of berth 2, only two of the three cranes can work at any one time on export containers.

Rule 8 is an example implementation of a delayed decision. A crane can only be re-allocated to export work once authorization has been obtained from a manager. The crane operator establishes in principle that the crane should be re-allocated. The shared boolean variable '_GetAuthorization' is then set to true. The crane operator has to leave the crane booth to obtain authorization from the manager and during this time the crane becomes non-operational. This is achieved by setting the shared boolean variable 'CraneOperational' to False. The time taken to obtain authorization is defined by the shared integer variable 'TimeToGetAuthorization' and is determined by the time of day. Hence, the variable 'TimeToGetAuthorization' is set by calling the Pascal function 'CalcFromCurrentTime' which returns the appropriate duration by checking the current simulation time. When control is returned to the simulation model from the expert system, if the '_GetAuthorization' boolean variable was set to 'True', a 'B' event is scheduled to occur after a time period equal to 'TimeToGetAuthorization'. Until this time is reached, the fact that the boolean variable 'CraneOperational' was set to 'False' ensures that the crane remains idle and prevents the process of authorization from re-occurring. On execution of the 'B' event, control is returned to the expert system and an attempt made to satisfy the goal 'AuthorizeCraneToExport' which is represented by rule number nine in figure 55. In rule nine, the manager simply verifies the same model data as the crane operator and therefore always gives authorization. A simple alternative is to introduce a random element to the crane operators decision rules, thereby ensuring that a mistake is occasionally made in seeking authorization from the manager. The manager would then overrule the initial decision taken by the crane operator.

Rules are classified into rule-sets to increase the 'legibility' of the knowledge-base and to improve expert system performance by limiting the search space. In the case of the 'CraneManager' rule-set, the 'Inherit' command has been used to define two other rule-sets which can be used if the inference engine cannot resolve the goal. The rule-sets correspond to each of the entity life cycles to which a manager has been allocated. As the number of rules increases and performance of the model degrades, the existing rule-sets can be sub-divided into smaller units.

When rules are added to the knowledge-base, errors are typically made. Syntax errors are trapped and reported by the part-compiler. Logical errors are often detected when a goal cannot be resolved, through unusual behaviour of the model, or through error messages generated by the simulation code library. The validation of the model is described further in section 5.6

5.5 DESIGN OF THE MAN/MACHINE INTERFACE



ES Knowledge-Base Filename : ess.exp_____

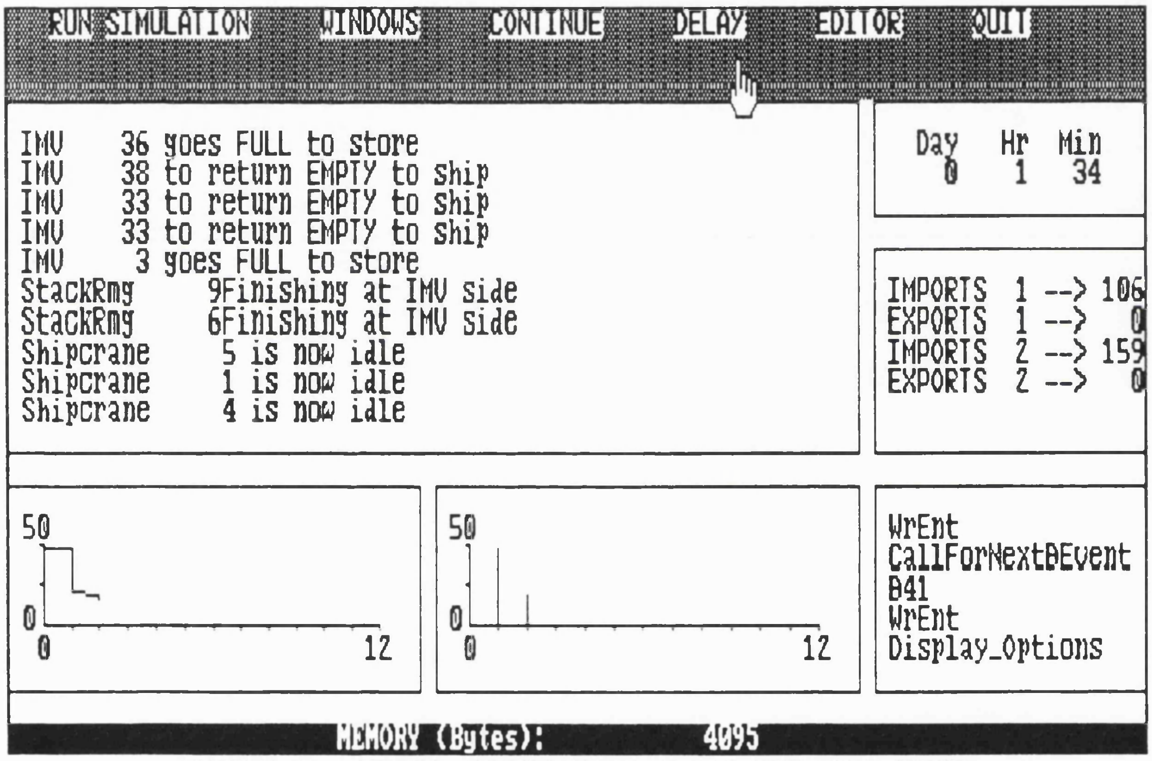
Run Length : 1000_

FIGURE 57 STARTUP MENU OPTIONS FOR THE PORT MODEL

A user friendly interface for the port model was seen as being essential in visually validating model output and providing a common front-end for the simulation and expert system components. The creation of the interface was greatly eased through the use of the 'Designer' program which enabled the display to be generated from an interactive user session. The basic screen designs were produced and the associated code generated in under an hour. Once the screens have been produced, the generated user interface has to be linked to the simulation model. This is achieved by replacing the Pascal 'Readln' and 'Writeln' commands with 'Designer' specific commands which re-route output into specific windows. The process undertaken in producing a model's user interface using 'Designer' is fully described in appendix E.

The interface is purely graphical with a set of user options displayed on the top line (see figure 57). The mouse is used to point at one of the options. Pressing the left mouse button results in either the display of a pull-down menu or a pop-up window. When the port program is first initiated, the modeller typically selects the 'Run simulation' option which result in the successive display of two pop-up windows. The first of these is used in specifying the desired duration of the simulation run, and the second for the selection of the expert system knowledge-base. Several knowledge-bases can be created for a single model, thus easing the process of experimentation.

The initiation of the simulation run activates the knowledge-base part compiler. Variable declarations are extracted and procedure or function calls identified. The rules declared in each of the rule-sets are then translated to reverse polish and optimised as was described in section 4.4.3. At this stage, any errors in syntax are reported to the modeller and an option displayed allowing him to load the text editor. The display is divided into three windows in which the appropriate text scrolls. the upper-most window lists the content of the knowledge-base with additional text being revealed during the scanning process. This ensures that errors are quickly identified as any syntax problems will be located in the last line of the window. The centre window lists the variables extracted from the knowledge-base. Any interruption in this process indicates an error in memory allocation either resulting from the repeated declaration of a variable name, or from the a lack of RAM. Such errors, and the status of the part-compiler are listed in the lower window.



```
IMV    36 goes FULL to store
IMV    38 to return EMPTY to ship
IMV    33 to return EMPTY to ship
IMV    33 to return EMPTY to ship
IMV     3 goes FULL to store
StackRmg     9Finishing at IMV side
StackRmg     6Finishing at IMV side
Shipcrane     5 is now idle
Shipcrane     1 is now idle
Shipcrane     4 is now idle
```

```
Day   Hr  Min
 0     1   34
```

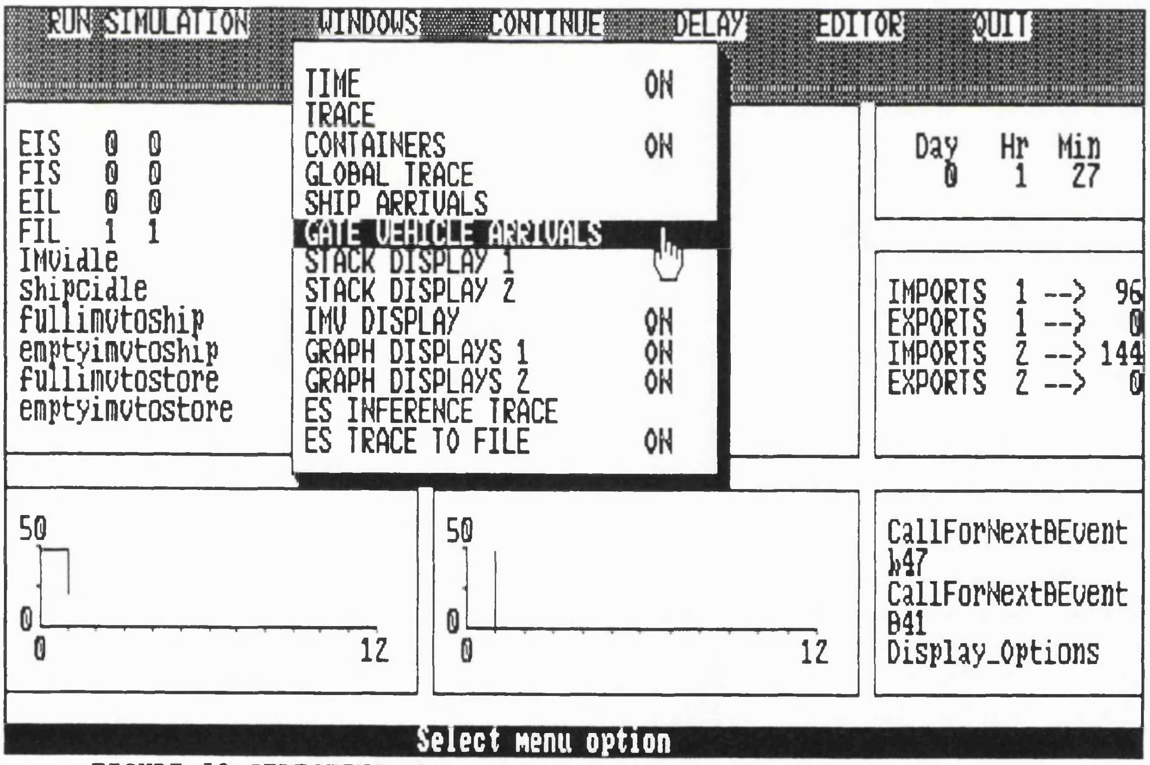
```
IMPORTS  1 --> 106
EXPORTS  1 -->   0
IMPORTS  2 --> 159
EXPORTS  2 -->   0
```

```
WrEnt
CallForNextBEvent
B41
WrEnt
Display_Options
```

```
MEMORY (Bytes):        4095
```

FIGURE 58 DEFAULT OUTPUT DISPLAY FOR THE PORT MODEL

If the part-compilation process ends without error, the simulation begins and the screen display changes to that shown in figure 58. The upper-left window is a general display for the status of the simulation. The two lower windows are graphical displays of queue lengths for arriving trucks. The current simulation time is shown in the upper right window. Immediately below this is shown the number of import containers which have been unloaded from the ships in berths one and two, and the corresponding number of export containers which have been placed in the holds. The window in the lower right hand corner is a trace of the C-Event procedures and goals that have been resolved by the expert system. The trace facility is a useful means of determining the location of errors which ESSIM does not report. The bottom line of the screen display is used for displaying user instructions.



```
EIS   0  0
FIS   0  0
EIL   0  0
FIL   1  1
IMVidle
ShipCidle
fullimvtoship
emptyimvtoship
fullimvtostore
emptyimvtostore
```

```
TIME                      ON
TRACE
CONTAINERS                ON
GLOBAL TRACE
SHIP ARRIVALS
GATE VEHICLE ARRIVALS
STACK DISPLAY 1
STACK DISPLAY 2
IMV DISPLAY               ON
GRAPH DISPLAYS 1          ON
GRAPH DISPLAYS 2          ON
ES INFERENCE TRACE
ES TRACE TO FILE          ON
```

```
Day   Hr   Min
 0     1    27

IMPORTS  1 -->  96
EXPORTS  1 -->   0
IMPORTS  2 --> 144
EXPORTS  2 -->   0

CallForNextBEvent
b47
CallForNextBEvent
B41
Display_Options
```

```
Select menu option
```

FIGURE 59 SELECTING ALTERNATIVE DISPLAYS FOR PORT MODEL OUTPUT

The modeller may chose to analyse the effectiveness of alternative port designs or management rules through a visual comparison of system behaviour. Graphical displays which attempt to depict the flow of materials are of general interest but cannot provide the required level of detail. The approach used in the port model was to make use of the window based displays to depict the model in terms of queue lengths, entity status, and knowledge-base traces. Pointing the mouse and clicking on the 'windows' menu option interrupts the simulation and results in the display of a pull-down menu (see figure 59). The status of the various options is displayed on the right hand side of the window. Menu entries permit the display of further windows for the analysis of output from the ship, gate vehicle, stack, and IMV cycles. Typical output is shown in appendix G. Two of the menu options relate to the expert system. The 'ES inference trace' menu option permits a trace produced by the expert system to be displayed and updated dynamically during the simulation run.

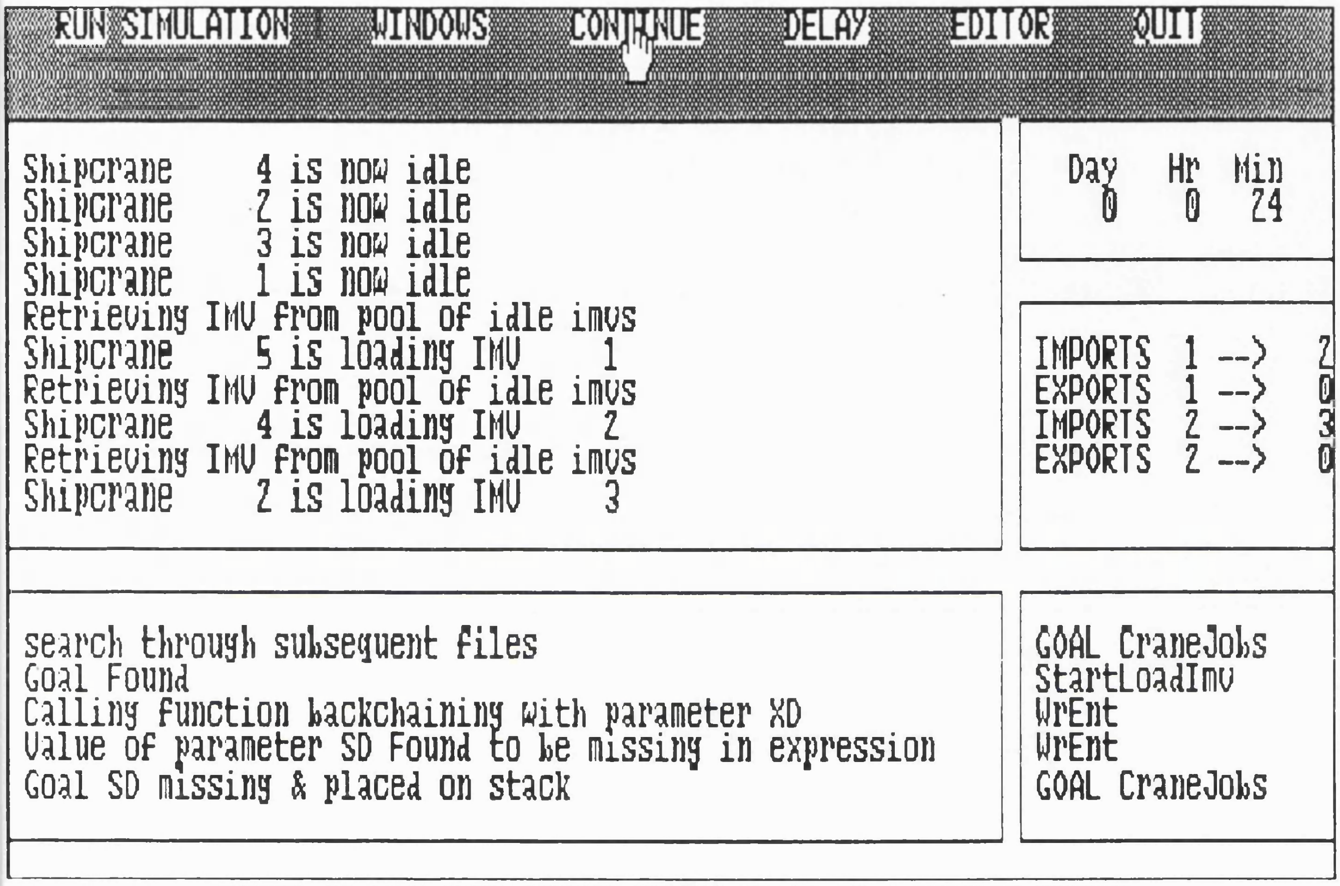
The 'ES trace to file' options is used in conjunction with the expert system 'Trace' command (section 4.4.1). A large amount of data is produced by the expert system trace. The 'ES trace to file' option was therefore provided to enable the user to switch the trace on and off during the simulation run.

The 'delay' option in figure 59 is used in slowing down the simulation in situations where output is displayed too rapidly. The 'editor' option permits a text editor to be used in modifying the expert system knowledge-base. With experience, minor modifications can be made to rules during the simulation run in assessing the effects of changes in policy.

5.6 MODEL VALIDATION

In creating the model of the automated container port, no real-world system was available for the purpose of comparison and validation. The model was consequently developed on an incremental basis such that system behaviour could be validated by experienced port managers at each stage of the implementation process. Tables relating to resources levels and activity status were displayed during the simulation run such that the port managers were able to visually identify abnormalities. A simulation output trace was also used in checking that scheduled events were occurring and that entities were correctly added to queues. A trace of procedure names enabled the location of fatal errors to be determined. Constant durations were also used in easing the process of output validation. Many logic errors were detected through fatal errors such as attempting to remove entities from empty queues.

Having created the standard three-phase model of the port, a new version was implemented using ESSIM and its associated expert system. The first stage consisted of re-creating the original model in terms of its apparent behaviour. Rules that had been defined in Pascal were extracted and re-created using ESSIM's knowledge-base syntax. This process highlighted the limitations of the first version of ESSIM, and so modifications were made to improve the functionality of the system. This ability to customise the expert system was naturally of great benefit. The original model was then used in ensuring the correctness of the ESSIM version.



```
Shipcrane     4 is now idle
Shipcrane     2 is now idle
Shipcrane     3 is now idle
Shipcrane     1 is now idle
Retrieving IMV from pool of idle imvs
Shipcrane     5 is loading IMV     1
Retrieving IMV from pool of idle imvs
Shipcrane     4 is loading IMV     2
Retrieving IMV from pool of idle imvs
Shipcrane     2 is loading IMV     3
```

```
Day   Hr  Min
 0     0   24
```

```
IMPORTS  1 -->  2
EXPORTS  1 -->  0
IMPORTS  2 -->  3
EXPORTS  2 -->  0
```

```
search through subsequent files
Goal Found
Calling function backchaining with parameter XD
Value of parameter SD Found to be missing in expression
Goal SD missing & placed on stack
```

```
GOAL CraneJobs
StartLoadImv
WrEnt
WrEnt
GOAL CraneJobs
```

FIGURE 60 REAL-TIME EXPERT SYSTEM TRACE IN A DISPLAY WINDOW

The ESSIM knowledge-base consists of individual rules which are defined in any order, and which may or may not be, in some way related. The expert system inference strategy follows a simple recursive backward-chaining pattern which in practice is difficult to trace given the lack of any visible sequential coding structure. The simplest means of verifying the logic applied by the expert system is to scrutinise the traces that are produced and to consequently confirm that the conclusions reached for each rule are correct, based on the input data being used. Having validated the ESSIM version of the model, subsequent modification of the knowledge-base through the modification of existing rules or the addition of further constraints was comparatively simple. Once again, inference traces were the simplest means of confirming the correctness of conclusions reached (see figure 60).

The process of model validation was eased following a reduction in the expected number of errors consequent to the application of strict standards in the structuring of code. For instance, the use of modular programming techniques permitted the isolation of blocks of code corresponding to each of the major entity life-cycles. The localisation of variables by direct association with specific modules minimised the risks of misusing data stored in other modules. The use of interfaces between modules also helped in formalising the permitted interaction of queues used in different life-cycles. Hence, modifications made to a life-cycle defined in one module were less likely to have an undesirable effect on another module (see section 5.3).

The isolation of management rules within a knowledge-base, which were previously embedded in pascal code, also eased the process of model validation. Previously, all simulation code had to be read in finding and then checking the coded logic. In the case of the ESSIM model, management rules are appropriately grouped and can easily be checked against stipulated management practices.

5.7 MODEL EXPERIMENTATION

One of the purposes in developing a simulation environment based on the use of an expert system was to ease the process of refining management rules through experimentation. Improved accuracy, maintainability, and adaptability were seen as some of the eventual goals of the research.

The ESSIM model of the container port was used as a basis for further experimentation through the enhancement and modification of management rules. Alterations to rules were also made in changing process durations and assessing the effect of changes in the number of entities within life cycles.

The complexity in implementing changes to expert system rules is largely dependent on the impact these have on the Pascal simulation code. Modifications to decision making rules which only involve the expert system are generally simple to implement. Conversely, changes which involve major modification to the life-cycle structure of the Pascal model are necessarily complex. In order to adequately compare the process of implementing changes in conventional three-phase simulation models and ESSIM, a number of experiments were staged which gradually increased in complexity. These are fully reported in the following sections.

5.7.1 Experimenting with rule parameters.

The easiest experiments to carry out consist of altering the parameters to existing rules. No modification to variables need to be made other than a possible re-classification of a variable from an integer to a real type. The simulation experiment can then be initiated without the need to re-compile the simulation code. The modeller can also repeat the experiment using different rule parameters without ever leaving the ESSIM program. ESSIM incorporates a text editor which permits the modeller to modify the interpreted expert system rule-base and then immediately commence a new model execution. It is also possible with a good knowledge of ESSIM, to interrupt a simulation run, modify rule parameters and re-commence model execution from the point of interruption.

OLD RULE

```
[1] (ImvsIdle = True) AND (ReturnImvsToIdle = True) ~ (ImvsIdle = False)
        IF (BerthedShip = False) AND (NumEmptyImvAtShip >= 5) ;
```

NEW RULE

```
[1] (ImvsIdle = True) AND (ReturnImvsToIdle = True) ~ (ImvsIdle = False)
        IF (BerthedShip = False) AND (NumEmptyImvAtShip > 0);
```

FIGURE 61 EXPERIMENTATION USING RULE PARAMETERS

The example in figure 61 specifies that IMVs should return to the central depot when five or more empty IMVs are waiting at a berth which is no longer occupied by a ship. The modified rule stipulates that IMVs should return to the central depot regardless of the number of idle IMVs waiting at the berth. This simple experiment is very easily conducted using the ESSIM expert system. In a conventional three-phase modelling environment the modeller would have hard-coded the rule within the Pascal model. In the case of a model which is substantial in size and spread over several code modules, the modeller would first have to locate the rule, re-compile the code module, re-link the modules into an executable and then recommence the simulation run. A conventional model could of course be designed to read data files which are loaded at run-time, but unlike ESSIM, the modeller would have to decide in advance the exact experiments which would be carried out such that the appropriate variables were defined within the data files.

5.7.2 Experimenting with variable values within rules.

Another class of experiment which was also performed consisted of altering characteristics of the model such as process durations and number of entities in a cycle. If the simulation model is appropriately structured, more major modifications can be made through the alteration of appropriate variable values. Three examples are given in figure 62. The first of these statements belongs to the 'ImvManager' rule-set and is used in defining the total number of IMVs that can be operational in the IMV cycle at any one time. The modeller can experiment with alternative upper limits on the number of IMVs by modifying the statement and re-starting the simulation. No compilation of code is required. Rule seven in the 'StackManager' ruleset defines the condition under which the gantry crane in a stack should move empty to the ship side of the storage area. The rule returns to the simulation model the activity duration. Rule seven stipulates that the gantry crane can move from the idle position to the ship side of the storage area in 40 seconds. A cheaper gantry crane can however be purchased which takes 80 seconds to cover the same distance. The ESSIM user can experiment with alternative durations without resorting to altering and re-compiling Pascal code. Hence the effect on the model of using the cheaper gantry cranes can easily be assessed. The modeller can also define the activity duration as being subject to the outcome of a decision rule. The third example consists of a definition of the number of stacks in the storage area. All data structures used in the Pascal model are defined using upper-limits which are set using variables. Such variables can be altered from within the expert system knowledge-base. Altering the number of stacks results in a physical change to the model in that appropriate queues are either added or removed according to the value associated with the 'NumberOfStores' variable. Once again, re-compilation is not required.

```
RULESET ImvManager (INHERIT ShipManager, StackManager) ;
..........
[*] NumberOfImvs = 100 ;
..........

RULESET Stackmanager ;
...........
[7] (MoveGantryToShipEmpty = True) AND (JobToBeDone = True)
          AND (_MoveGantryToLandSide = False) AND (TimeToMoveToShipEmpty = 40)
          IF (_MoveGantryToShipSide = True) AND (NextShipJobIsAnImport = True) ;
...........
[*] NumberOfStores = 10 ; {Number of stores is 10}
...........
```

FIGURE 62 EXPERIMENTING WITH RULES

5.7.3 Experimenting with rule structures.

The types of experiments which we have so far examined have simply consisted of alterations to variable values or rule parameters which could have been accomplished, although with relatively less ease and neatness, using a conventional modelling environment. The benefits of the ESSIM approach to modelling really emerge when one considers modifications of greater complexity which involve the replacement of existing decision rules or the introduction of new rules. In this context, the benefit of the ESSIM approach are centred on the fact that decision rules are isolated from the rest of the model and encapsulated within an expert system knowledge-base. Therefore, many experiments based on decision rules can be conducted without recourse to often substantive volumes of detailed low-level code relating to the description of the physical components of the model and their interaction.

The modeller can alter the structure of a rule or group of rules without having to modify the Pascal code so long as the changes are limited to one of the following:

1. Simplifying a rule by removing some of the variables.
2. Altering the combination of operators used.
3. Adding additional external variables as conditions to existing rules, on the condition that these variables are already declared within the expert system knowledge-base.
4. Adding local variables which are unique to the expert system to existing decision rules.
5. Creating new decision rules using local or external variables which have already been defined within the expert system knowledge-base.

OLD RULE:

```
[8] (_ChangeACraneToExport = True) AND (GetAuthorization = True) AND
    (TimeToGetAuthorization = CalcFromCurrentTime) AND (CraneOperational = False) ~
    (_ChangeACraneToExport = False) AND (GetAuthorization = False)
  IF    ((CurrentShipBerth = 1) AND (NumCranesOnImports > 0))
  OR    ((CurrentShipBerth = 2) AND (NumCranesOnImports > 1))
  AND  (NumImportsRemaining = 0) AND (NumTotalImportJobs > 0) ;
```

NEW RULES:

```
[8] (ChangeACraneToExports = True) AND (GetAuthorization = True) AND
      (TimeToGetAuthorization = CalcFromCurrentTime) AND (CraneOperational = False)
  IF    (CurrentShipBerth = 1) AND ((NumImportsremaining < (NumTotalImportJobs / 2))
  AND (NumCranesOnImports = 2)) OR ((NumImportsRemaining = 0)
  AND (NumCranesOnImports = 1)) ;

[9] (ChangeACraneToExports = True) AND (GetAuthorization = True) AND
      (TimeToGetAuthorization = CalcFromCurrentTime) AND (CraneOperational = False) ~
      (_ChangeACraneToExport = False) AND (GetAuthorization = False)
  IF    (CurrentShipBerth = 2) AND ((NumImportsRemaining < (2 * NumTotalImportJobs / 3))
  AND (NumImportsRemaining > (NumTotalImportJobs / 3)) AND (NumImportsRemaining > 0)
  AND (NumCranesOnImports = 2)) OR ((NumImportsRemaining = 0)
  AND (NumCranesOnImports = 1)) ;
```

FIGURE 63 ALTERING RULE LOGIC

The modeller can modify groups of rules without corresponding alteration of the Pascal code on the condition that the defined goals can still be resolved. In some cases, the expert system returns additional values such as process durations that the simulation model requires. The modeller must ensure that such values are not omitted. On the condition that no new shared variables are defined, or pascal functions added, re-compilation of code can be avoided and the modeller need not access the Pascal model code. Certain abnormalities such as syntax errors in the knowledge-base are reported to the modeller immediately prior to the execution of the model. Other inconsistencies, such as the inability to resolve a goal are reported during model execution. The modeller is then able to edit the knowledge-base or re-run the model in 'trace' mode which reports on the results of each of the expert system's inference steps.

The example rule given in figure 63 is taken from the 'CraneManager' ruleset which was described in section 5.4. A complete listing of the rules can also be found in figure 55. The rule operates in conjunction with the other rules in the 'CraneManager' Ruleset and originally stipulated that all import containers should be unloaded from the holds of the ship prior to loading the consignment of export containers. In the case of the second berth which has three cranes, only two of the cranes can operate on export containers at any one time. An alternative scenario that was investigated was the possibility of allowing import containers to be unloaded from the ship whilst other cranes carried out the reverse operation of loading export containers from arriving IMVs. In this case, all three cranes working at berth 2 were allowed to operate simultaneously on loading ships with export containers. At the start of the simulation, all cranes are allocated to imports. The number of cranes subsequently re-allocated to exports is in proportion to the remaining workload. In the case of the berth with two cranes, once half the ships consignment of import containers remains, one of the cranes is re-allocated to exports and the loading process begins. Once all import containers have been unloaded from the ship, the remaining crane changes to working on exports. Two additional rules were introduced into the 'CraneManager' ruleset in implementing the alternative scenario and are shown in figure 63. No modification of the simulation code was required and the experiment was consequently carried out without having to re-compile any of the modules or indeed accessing any of the Pascal model code.

Model changes may be initiated that require both the introduction of further rules and associated modifications to the simulation code. For instance, the modeller may wish to add further decision rules to a knowledge-base ruleset, the conditional statements of which contain simulation model variables which have not previously been accessed by the expert system. Such changes are relatively simple to implement. The modeller must declare the necessary Pascal variables relating to the required model data as being of type 'Public'. This permits the data values to be accessed from other Pascal code modules and the expert system. The Pascal module containing the variable declarations is then re-compiled. By adding the variable name to the list of variables declared in the expert system, the modeller may then make use of the new variable values within rules.

A more complex task is the introduction of new entity cycles within the simulation model and the corresponding addition of model logic through the use of further expert system rules. Modifying the structure of a simulation model is a testing task and was highlighted as a limitation of existing modelling techniques in section 2.2.4.

A complex experiment was carried out to evaluate the consequence of having to introduce detailed changes into the ESSIM simulation model of the Port and corresponding rule-sets. In the original port model the lay-out of which was depicted diagrammatically in figure 43, Rail Mounted Gantry cranes are used in transporting containers within the Stacks in the port storage area.

RMGs have the limited ability to load and off-load waiting Internal Movement Vehicles (IMVs) and Gate Vehicles (GVs), moving containers to and from their storage positions. Export containers allocated to specific ships and import containers are sent at random to one of the stacks within the storage area. Whereas the workload at each of the stacks is roughly balanced, the spread of containers may be inefficient if the majority of containers for one ship end-up in one stack and those for another ship in another stack. The RMGs can only handle one container at a time and so the more evenly spread across the storage stacks are the containers for a specific ship, the more efficient is the ship loading process as the movement of containers may then take place in parallel. The new experiment to which the ESSIM port model was subjected, consisted in the addition of further entity cycles with the aim of redressing the problem of optimising the spread of containers between stack storage areas. A new type of vehicle, known as a lateral Movement Vehicle (LMV) was added to the model. An LMV operates in each of the storage area stacks and has the ability to shift containers from one stack to either of the two immediately adjacent stacks.

```
VAR     QStackVehicleIdle : ADS OF ARRAY[1..MaxStores] OF QUEUE ;
        e2 : Entity ;

QStackVehicleIdle := ALLMQQ(Wrd(MaxStores*2));        {Allocate RAM}
e2 := StackVehicle ;
FOR i := 1 TO NumberOfStores DO
BEGIN
      MakeQ(QStackVehicleIdle^[i],    'QStackVehicleIdle',e2);
      FillQueue(e2,1,QStackVehicleIdle^[i]);
      e2 := e2^.next ;
END;
```

FIGURE 64 DEFINING QUEUE STRUCTURES FOR LMVs

Queues relating to each of the LMVs were defined within the Stack module of the Pascal simulation program. A Corresponding C and B event were also added describing the process of shifting containers between adjacent stacks. The first step was to define the queue structures and initialise these by requesting the necessary memory and filling the queues. The relevant code is show in figure 64.

```
PROCEDURE START_LMV_MOVE ;
BEGIN
     Write_Window(2,'Start_Lmv_Move');
     FOR i := 1 TO NumberOfStores DO
     BEGIN
          FOR ShipCode := 1 TO NumberOfShips DO
          IF BalanceStacks THEN
          BEGIN
               IF _TransfExpToRhtStack THEN TransferExportToRightStack ;
               IF _TransfExpToLftStack THEN TransferExportToLeftStack ;
               IF _TransfImpToRhtStack THEN TransferImportToRightStack ;
               IF _TransfImpToLftStack THEN TransferImportToLeftStack ;
          END;
     END;
END;


PROCEDURE END_LMV_MOVE ;
BEGIN
     Write_Window(2,'End_Lmv_Move');
     IF GlobTrace THEN
     BEGIN
          OutTxt := Wrent(Current);
          CONCAT(OutTxt,'Stack Vehicle is now idle');
          Write_Window(3,OutTxt);
     END;
     Addto(Back,QStackVehicleIdle^[Current^.attr],Current);
END;
```

FIGURE 65 C & B EVENT ROUTINE FOR THE LMV CYCLE

The C and B type events that were added to the stack module are shown in figure 65. 'BalanceStacks' is a Pascal function from which the call to the expert system knowledge-base is initiated. Actions that need to be taken are then passed back to the simulation model by the expert system and are described in the form of boolean shared variables (e.g._TransfExpToRhtStack).

```
FUNCTION BalanceStacks : Boolean ;
BEGIN
     Write_Window(2,'GOAL   BalanceStacks');
     ResultAddr  :=   GOAL('StackManager','BalanceStacks');
     BalanceStacks := ResultAddr^ ;
END;
```

FIGURE 66 GOAL CALL TO THE EXPERT SYSTEM

The content of the 'BalanceStacks' function is shown in figure 66. The 'Goal' function is used to call the expert system 'StackManager' ruleset and check whether the appropriate LMV should be activated. The return value is in the form of an address which is read and associated with the boolean function return value. The corresponding expert system rules are listed in appendix C. An addition ruleset known as 'LMVmanager' was created which is linked with the 'StackManager' ruleset through the use of the 'Inherit' command (see section 4.4.1). The use of a separate ruleset eases the interpretation of the knowledge-base rules whilst retaining the association between LMVs and Stacks.

As was demonstrated, the actions necessary in modifying the simulation and expert system code are relatively simple. Having initiated the changes, the modeller can carry out further experimentation by altering the expert system rules. These further experiments may no longer require of the modeller to access the Pascal code.

Had the modeller been introducing the LMV cycles into a conventional three-phase mode, the complexity of the work would have been aggravated as a consequence of the lack of code modularity. Code modularity is introduced at two levels within the ESSIM port model. The Pascal code is itself broken down into individual modules representing each of the major entity cycles. Secondly, the expert system knowledge-base is itself a code module into which decision rules are isolated. Additionally, what proves to be of significant benefit in the ESSIM model is the fact that the process of introducing changes follows a strict sequence with the layout of the code strictly controlled through the segregation of logic between simulation model and expert system knowledge base. This has a clear impact on the maintainability of the model and the subsequent adaptability in altering rule conditions during the experimentation phase.

5.8 THE ALTERNATIVE PORT MODELS.

Prior to the implementations of the ESSIM version of the port model, a simplified model was developed solely using Pascal code. Rules specifying the conditions under which activities should start were defined using IF-THEN conditions and boolean statements. These rules were specified as part of each of the C event procedures. The example in figure 67 shows the structure of one such typical procedure.

The ESSIM port model was then implemented by first extracting the rules from the pascal code and replacing these with calls to the expert system. The isolated rules were added to the expert system knowledge-base and later enhanced through the specification of more detailed conditions. Comparison of the two models indicated that modifying the expert system knowledge-base was in many cases simpler than altering Pascal code. These comparisons are discussed in section 6.3.

```
PROCEDURE C1 ;
Begin
   IF Condition THEN
   Begin
    ..........
   End;
End;
```

```
FUNCTION Decision: ReturnVal ;
Begin
 ............
End;

PROCEDURE C1;
Begin
  IF Decision THEN
  Begin
   .............
  End;
End;
```

Figure 67 Representing decision tasks in Pascal

The conventional Pascal based model of the port and the ESSIM version are based on different principles. The purely Pascal based version permits the development of significantly sized models but provides the user with limited scope for the specification of the conditions associated with the start of an activity. The ESSIM port model rectifies the balance by providing a means of specifying goals which can be resolved using a bespoke expert system shell. Some researchers including Alty[1984], have voiced doubt as to the effectiveness of the expert system approach, arguing that a similar level of functionality could be achieved using a conventional 3rd generation language. A third version of the port model was consequently developed with the aim of providing a similar level of functionality as the ESSIM model, but coded entirely in Pascal. The work involved in developing this further example was felt to be justified given that a more detailed comparison of the approaches could then be made.

The principle applied in developing the new model was to replace defined goals by Pascal functions which would return values to the simulation model in a similar way to the expert system. These functions were then placed together in a single module, thus providing a similar logical separation as was achieved between the Pascal model and expert system knowledge-base. The content of the ESSIM knowledge-base can be found in appendix B. The Code for the Pascal version of the expert system knowledge-base is included in Appendix D. The variables used in the Pascal implementation are the same as those that were used in the expert system knowledge-base. This was possible because of the implementation of Pascal data types as part of the knowledge-base syntax. The Conversion of rules to Pascal was in itself a relatively simple task given that ESSIM's production rule syntax was replaced by the Pascal IF-THEN-ELSE instruction. The sequential nature of code execution in Pascal represented a far greater problem. The rules were consequently sequenced such that the premises to each rule could always be resolved. The lack of an equivalent to ESSIM's 'Inherit' command also meant that repetitively used rules had to either be duplicated across the code or placed individually in functions.

5.8.1 EXPERIMENTING WITH THE MODELS.

The port model developed purely in Pascal using the eLSE routines will be referred to as the 'Original model'. The other two versions will be described as the ESSIM and 'Function' based models respectively.

The experiments described in section 5.7 were repeated using the other model versions in order to identify the merits and limitations of each of the approaches. The experiences in developing each of the models and the results of the model comparisons were then used in assessing the thesis proposition. The conclusions drawn from the experiments carried out using each of the three modelling environments are reported in the following sections.

Comparing the adaptability of the model representations:

It has been proposed that combining simulation and expert system techniques could provide the modeller with a simulation environment which is better suited to the task of experimenting with alternative operational policies. Evidence was sought by carrying out a number of model experiments on each of the three versions of the port model. These experiments can broadly be described as follows: 1) Changes to rule parameters. 2) Changes in operational policies reflected through the introduction of new or modified rules. 3) Modifications to the model representation of the real-world involving the introduction of a new activity cycle and related operational policies.

In carrying out the experiments discussed in section 5.7, it was found that the port model written using ESSIM was generally the easiest to modify for the purpose of evaluating alternative operational policies. There are several reasons for this, but the single most predominant factor is that in ESSIM, the decision rules that form part of operational policies are defined in a highly structured fashion.

In the original Pascal model, operational policies were represented as multi-level conditional statements which preceded each of the "C" events. The rules which in the real-world would have been applied by different decision makers were combined and sequenced such that all eventualities could be considered. Sequencing decision rules in this way made it very difficult for the modeller to introduce changes which reflected alternative operational policies.

In the ESSIM version of the port model, a statement was inserted prior to each "C" event which effectively transferred control to the expert system module. Within the expert system knowledge-base, decision rules were grouped into rule-sets which were named according to the job function of the decision maker. Rule-sets were then linked together using "Inheritance", in order to either reflect hierarchical management structures or situations in which multiple decision makers would act together. Operational policies of this nature were termed as "Cooperative decision making" and discussed in section 4.4.2. Deducing conclusions from the defined rules was achieved using an inference engine which eliminated the need to pre-sequence and interlink individual rules. The experiments carried out in section 5.6 showed how straight forward it could be to alter operational policies. Firstly, the decision rules were isolated from the rest of the model. Secondly, altering an operational policy could be as simple as identifying the appropriate rule-sets and adding or replacing rules. In contrast, implementing the same changes to the conventional Pascal model could require the modeller to alter the sequence of rules in a multi-level conditional statements which in some cases could extend to over three pages of code.

The "Function" based version of the port model was built in an attempt to overcome the limitations of the conventional Pascal model. Pascal functions were used in an attempt to re-create the functionality of the rule-sets used in ESSIM. The "Knowledge-base" written using Pascal functions is listed in Appendix D. The approach failed in three key respects. 1) The language syntax was unnecessarily complex. 2) The need for complex multi-level conditional statements could not be eliminated. 3) Each defined function could require as many "Begin" and "End" statements as there were actual rules. In many cases it was simpler to repeat a rule rather than enclose it within a function statement.

The third class of model experiment to which the three versions of the port model were subjected, consisted in introducing a new activity cycle and related operational policies. The complexity of carrying out such drastic model changes was of interest, as ESSIM was only designed for the purpose of experimentation with operational policies. It was concluded in section 5.7.3 that ESSIM simplified in only some small respects the introduction of the additional "Lateral Movement Vehicle" entity cycle. The main benefit was that the separation of operational policies from the rest of the model representation acted as an additional level of code modularity. The introduction of further modularity forced the modeller to take a more structured approach to the alteration of the model. The first step was to create the Pascal representation of the activity cycle. Only once this had been completed would the modeller turn to the definition of the operational policies within the expert system knowledge-base.

Comparing the accuracy of the model representations:

The ESSIM environment is used for the purpose of developing models of real-world systems in which complex operational policies need to be represented. The ESSIM approach is meant to support this task by providing a modelling technique which results in a better representation of the real-world problem.

The experiments which were reported in section 5.7 highlighted the fact that the problem of representational accuracy is in fact very closely related to that of model adaptability. As was reported in the previous section, the experiments served to demonstrate that ESSIM decision rules were relatively adaptable because the ESSIM knowledge-base was a better representation of operational policies than the multi-level conditional statements used in the equivalent Pascal model.

The ESSIM development environment was designed specifically to address the issue of representing complex operational policies which could in turn necessitate the modelling of "Cooperative decision making". The conventional Pascal modelling approach catered for the representation of operational policies only by providing a general purpose 3rd generation programming language. The key differences between ESSIM and the Pascal and Function based models will now be discussed in turn.

The first fundamental structural difference between the modelling approaches is that ESSIM groups decision rules into rule-sets according to the job function of the decision maker. In the Pascal model, no distinction is made between the rules applied by one decision maker from another.

The second major difference is that ESSIM attempts to mimic the way real-world decision makers can act together in instigating an operational policy. The term, "Cooperative decision making" has been used in this thesis to reference such an approach. The rules defined within a given rule-set may be a representation of the totality of a decision makers knowledge in the context of a given job function. In practice, the decision maker's "Knowledge" may prove insufficient for the purpose of resolving a given problem. In the same way, the rules contained within a rule-set may prove inadequate during an attempt to resolve a goal. In the real-world, the decision maker may consult another decision maker and thus bring into play a further base of operational expertise. In ESSIM, "Inheritance" is used to bring together otherwise disassociated rule-sets and thus represent this interaction between decision makers.

There are other ways in which Inheritance can be used to represent real-world situations. For instance, a rule-set may contain sufficient information for a given goal to be resolved. However, in the real-world a manager may oversee the decisions taken by a decision maker and over-rule or influence the course of actions. In ESSIM, the rule-set associated with the decision maker can be linked using Inheritance to the manager's rule-set. Under the normal course of events, a given goal would be resolved without the inference engine scanning the manager's rule-set. In order to represent the manager's influence within the ESSIM model, a sub-goal can be associated with the main goal. Once the main goal has been resolved, the sub-goal is triggered. The sub-goal also needs to be resolved before control returns to the simulation model. The sub-goal can only be resolved by a set of rules defined by the manager, the purpose of which is to validate and possibly modify the intended actions of the decision maker.

Comparing the maintainability of the model representations:

The maintainability of a model is related to the ease with which changes that have occurred in the real-world system can be reflected within existing code. The maintainability of the model is of particular importance in situations where changes are expected in the real-world environment during the life of the model. Such changes may encompass modifications to physical aspects of the real-world such as plant lay-outs, or may simply consist in changes to operational policies.

The key differences between the ESSIM model and the two alternative Pascal models in the context of code maintainability, is that ESSIM places greater emphasis on modularity and the representation of operational policies.

The concept of modularity extends to several of the ESSIM components. At the broadest level, ESSIM introduces modularity by splitting the representation of operational policies from the rest of the model. Changes in operational policy can then be introduced with potentially little or no effect on the simulation model component. Conversely, some situations permit modifications to be made to physical aspects of the model without effecting the expert system knowledge-base.

The expert system knowledge-base is also based on a modular structure, with rule-sets being used to encapsulate the decision rules applied by each decision maker. Once again, changes may be made to the rules applied by one decision maker without there being any parallel requirement to modify the rules applied by another decision maker. In the context of the Pascal model, operational policies are stipulated in the form of multi-level conditional statements and may consist of decision rules applied by several decision makers. Making model changes that reflect long-term modifications in operational policies may consequently be more complex to introduce.

The simulation model component of ESSIM was also divided into separate modules. Rather than create a single block of code, a modular version of the Pascal programming language was used so that the individual model activity cycles could be isolated from each other with appropriate interfaces defining their interaction. The introduction of code modularity to Pascal based three-phase simulation modelling was found to be of benefit in the context of code maintenance as the overall structure of the model was improved.

The experiment detailed in section 5.7.3. sought to evaluate the impact of introducing a new activity cycle and related operational policies to the existing versions of the port model. This experiment required modifications to be made to both the ESSIM simulation module and expert system knowledge-base. As was discussed earlier in this section, the use of a modular construct, allowed the introduction of a major model change using a highly structured approach. The first step consisted in the creation of a new simulation module containing the code representation for the "Lateral Movement Vehicle" (LMV).

This new module was then interfaced to the rest of the model. The final step was to create the appropriate expert system rule-sets which were to represent the operational policies relating to the LMV.

To summarise, the benefits and limitations of each of the model representations are given in figure 68.

ORIGINAL MODEL:

BENEFITS

Fastest code execution.
Single language syntax.

LIMITATIONS

Operational policies represented as multi-level conditional statements.
Difficult to 'read' the code used to represent the operational policy.
May be difficult to alter the code relating to an operational policy.
Difficult to code rules which span across multiple activity cycles.

ESSIM MODEL:

BENEFITS

'Rule-sets' permit operational policies to be defined using a modular construct.
'Inheritance permits the representation of 'Cooperative decision making'.
Incremental development of logic is easier to achieve.
Modelling environment well suited for experimentation with operational policies.
The ESSIM expert system provides a 'better' representation of operational policies.
The expert system was relatively easier to use than the Pascal representation.

LIMITATIONS

Slowest code execution.
Expert System syntax is limited.
For simple groups of rules, the expert system is an overhead.

FUNCTION BASED MODEL:

BENEFITS

Fast code execution.
Implemented entirely in Pascal.
Single module used for the specification of decision rules.

LIMITATIONS

Difficult to code decision rules.
Hard to alter decision rules.
Need to compile and link code.
Additional variables required.

Figure 68 Comparision of modelling techniques.

5.9 CONCLUDING THOUGHTS ON THE ESSIM MODULES

5.9.1 OBSERVATIONS ON ESSIM'S SIMULATION MODULE.

The template which is used for developing Pascal models for ESSIM was specifically designed for use in a modular coding environment. The development of the port model highlighted the benefits of adopting a structured modular approach. The port model is a realistic replica of a potential real-world environment but is by no means an exceptionally complex and detailed representation. The port model consists of approximately 3000 line of code which points to the need for a modular coding approach. The use of modules was found to ease the development of the port model by allowing the implementation process to be broken down into the creation of a series of sub-models each representing one of the major entity cycles. The use of interfaces between modules was seen as a means of formalising the interaction between entity cycles through common queues. ESSIM's modular construct supports the development of a model by a team of programmer working simultaneously on the implementation of each of the entity cycles. Once the model has been coded, the modular approach is found to simplify the validation and correction of the models behaviour by easing the process of identifying the location of logic errors. Experimentation with the model is also simplified because of the greater ease with which model changes can be implemented. Finally, ESSIM's modular approach permits the creation of models which exceed the limit of 64K on code size imposed by the DOS operating system (In ESSIM, each module can be 64K in size, with FAR addressing being used to extend the addressable memory to 1Mb).

The Pascal simulation routines provided with ESSIM are based on those provided in the eLSE simulation library (Chew[1986]). Modifications were made to reduce the size of the library and improve the performance of the routines. In particular, the use of addresses rather than numbered B event routines was found to both improve the legibility of the code and simplify the structure of the simulation executive.

The provision of dynamic displays is a particularly useful feature of ESSIM models, permitting a modeller to gain some insight into the potential behaviour of the real-world system without having to necessarily resort to output analysis techniques. These dynamic displays are created using a library of screen handling functions which are used by the programmer during the development process. The graphics routines, coupled with the 'Designer' program were found to reduce the time scale required to complete the model by limiting the effort required in coding the user interface. In the case of the port model, the dynamic output displays were found to ease the process of locating errors in the simulation code and provided a useful focal point in discussing the behaviour of the model with port management. During the experimentation process, the output displays provide the modeller and potentially the actual managers with an easy to understand summary of the status of queues and processes in each of the ports constituent entity cycles.

The ESSIM approach to modelling has some minor limitations. In particular, the use of a modular approach results in a processing overhead during the compiling and linking of the code. For the larger modules, the process of creating an executable image can take up to 10 minutes. However, when changes are made to a single module, other modules need not be re-compiled. The error reporting capabilities of the Microsoft compiler are also fairly crude compared to the Turbo Pascal compiler. The use of a third generation language also results in some inconvenience given the additional complexity of the language syntax over a bespoke modelling tool. Conversely, the use of Pascal does confer some benefit resulting from the additional flexibility conveyed by a general purpose language. This is particularly apparent when considering the range of data structures available.

The development overhead in using a third generation language in specifying model logic could be overcome using a similar approach to current CASE (Computer Aided Software Engineering) tools for database design. The programmer specifies the model design by using a graphical drawing tool to create an Activity Cycle Diagram. The design is validated in real-time by rules which, for instance, check that queues are always separated by activities. Queue names and activity durations are specified by using a mouse to select the appropriate screen icons. The simulation model code is then generated by following the basic principles that permit the translation of ACDs into Pascal code.

ESSIM as an environment is a relatively complex system which would benefit from a more powerful development environment. A mainframe based system, whilst potentially improving performance, would not provide the degree of flexibility in terms of windowed and graphical output as is possible with the current generation of PCs. However, workstations which provide the benefits of both processing power and enhanced graphical output would resolve many of the limitations of the current ESSIM system in terms of execution speed and memory capacity.

Finally, an improved means of interfacing to ESSIM's expert system could be of benefit. Presently, the expert system returns to the Pascal model values associated with solution to goals through intermediate Pascal variables. No means currently exists to ensure that all expected values are returned. This results in additional validation work to ensure that the model behaves as expected. A potential means of overcoming this problem would be to associate with each goal, a list of variables names through which goal results are returned. The simulation model would stop and a warning given if a return value was found to be missing.

BENEFITS

Model template specifically developed for modular programming.
Use of modules necessary for large simulation models.
Interface between modules formalises the interaction between cycles.
Memory addresses used for the activation of B procedures.
Library of graphics routines for the creation of dynamic displays.

LIMITATIONS

Modular approach results in a processing overhead.
The use of a 3GL reduces productivity during model creation.

Figure 69 ESSIM's Simulation module.

The benefits and limitations relating to ESSIM's simulation module are summarised in figure 69.

5.9.2 OBSERVATIONS ON ESSIM'S EXPERT SYSTEM.

The development of the ESSIM expert system was a considerable task given the need for close integration with the compiled simulation code.

Originally, the use of a commercial off-the-shelf product had been envisaged which would have considerably simplified the creation of the modelling environment. If an existing product had been used, the inability to customise the expert system would have limited the functionality of the environment and reduced the benefits that ESSIM confers over conventional modelling techniques.

The ESSIM expert system is customised for simulation modelling. The key differences are as follows:

1) The ESSIM expert system supports the definition of local and "Public" variables. Public variables are variables which can be shared with other programs, in this case a Pascal simulation program. This sharing of data was an essential pre-requisite to the development of models involving a simulation model and an expert system. The integration of the ESSIM expert system with the Pascal language is not limited to the sharing of variables, but also extends to the ability to activate Pascal procedures and functions. C and Fortran routines can also be called through the use of intermediate Pascal functions. During the development of the port model, this feature was found to be particularly useful as arithmetic computation is more effectively carried out using a procedural language.

2) Rule-sets are used to group decision rules according to the function of the decision maker. The use of rule-sets brought about a number of fundamental changes to the way in which operational policies are represented within a simulation model. In conventional Pascal models, decision rules applied by different decision makers are linked together as part of a multi-level

conditional statement. In ESSIM, the use of rule-sets permit a far more structured definition of operational policies and hence, supports the modeller in building complex representations. Chapter three highlighted the complexities of representing operational policies. The new term, "Cooperative decision making" was then introduced as a means of describing operational policies which are enacted through the participation of two or more decision makers. The ESSIM expert system uses the concept of "Inheritance" to support the modelling of cooperative decision making.

3) "Inheritance" is a technique used in the ESSIM expert system knowledge-base to link together otherwise unrelated sets of rules relating to each of the decision makers. In section 5.8.1, we saw that Inheritance could be used as a simple method of representing hierarchical management structures. There are other interesting benefits to using Inheritance. For instance, an operational policy may require the involvement of more than one decision maker, each represented by a different rule-set and each responsible for the management of activities in different parts of the real-world system. In conventional three-phase discrete event modelling, the modeller is encouraged to represent the real-world system as individual but nevertheless interlinked activity cycles. In ESSIM, the modeller is encouraged to represent operational policies as individual but nevertheless interlinked rule-sets which together may span across multiple activity cycles.

There are other peripheral benefits to the ESSIM expert system. For instance, in the context of ESSIM, the simulation model typically submits goals to the expert system on an almost continuous basis. Consequently, the expert system had to be capable of resolving goals within a short time delay if the simulation is to operate within real-time. The implementation of rule-sets reduced the search space in resolving goals submitted by the simulation model. This dramatically improved the performance of the expert system.

Much time was devoted to the development of the expert system. Nevertheless, some weaknesses exist. For instance, Pascal procedures and functions can be activated from the knowledge-base but parameters can only be passed by creating common variables. The integration of Pascal routines into knowledge-base rules would have been simplified had it been possible to use the Pascal language syntax for the passing of parameters. More generally, comprehensive error reporting would have helped in the detection of some errors which manifested themselves in unusual ways.

Finally, it must be said that the original aim was to produce an expert system which used a very simple syntax. In the context of the modelling of operational policies, a simple syntax helps greatly in understanding the effect of the constituent decision rules. The need to create an effective interface with the Pascal language and the gradual addition of further functionality to the expert system resulted in a syntax considerably more complex than had originally been intended. A revision of the language syntax would partially alleviate the problem. The use of a Natural Language Programming (NLP) approach could also be considered.

The benefits and limitations of ESSIM's expert system module are summarised in figure 70.

BENEFITS

Expert system customised for simulation.

Expert system highly modularised using 'Rule-sets'

'Inheritance' used to model 'Cooperative Decision Making'

Decision rules may pan across multiple activity cycles.

The use of rule-sets limits the search space during the inference process.

The expert system is highly integrated with the Pascal language.

LIMITATIONS

Fewer functions available than in commercial expert systems.

The expert system syntax is more complex than originally intended.

Figure 70 ESSIM's expert system module.

5.9.3 BENEFITS AND LIMITATIONS OF THE USER INTERFACE.

ESSIM's user interface development tools were felt to be of significant benefit in the implementation of the port model and the subsequent interpretation of the output produced. Models produced using ESSIM are intended to produce output which can be scrutinised at run-time by managers, who, with the assistance of the modeller may evaluate the effectiveness of alternative operating procedures. Consequently, the displays must provide an adequate means of visualising the changes in status through time of the real-world system being modelled.

The creation of a user friendly interface which provides facilities for the graphical representation of model output is a time consuming and complex process. The 'Designer' interface generator was found to be useful means of creating an initial lay-out for the menus and windows. The benefit of using 'Designer' was that the interface could be created interactively which permitted the visualisation of the screen design without the need to re-compile code when changes were made.

At the programming level, the provision of a library of window and graph manipulation routines was found to aid the modeller by removing the need to consider the low-level screen manipulation code normally associated with the development of applications for PCs. Combined with the use of 'Designer', the routines help to minimize the overhead of controlling screen output.

As far as the modeller is concerned, the simulation models developed using ESSIM and exemplified by the port model, are easy to use and effective as a means of communicating information. In particular, the concurrent updating of windows enables different parts of a model to be displayed simultaneously. The use of the mouse was also found to simplify the selection of menu options and was noted as being particularly useful in the context of ESSIM models in that the modeller typically continuously switches between alternative displays.

Whereas ESSIM eases the process of developing an effective user interface, there remains a significant overhead in code writing and the subsequent validation of output. The use of batch runs in which output is restricted to printed text remains a cruder but simpler means of producing a trace of a simulation run.

A concept which extended beyond the scope of this thesis and yet would have been interesting to explore is the possibility of using a process referred to as 'Interactive Programming' in specifying the screen design and the associated simulation code. The principles applied in creating 'Designer' could have been taken a step further by providing a means of generating simulation pseudo-code from an ACD design. This code could then have been integrated into the windowing environment and tested on an interpretive basis. Once satisfied with the final prototype, the associated Pascal code could then have been generated.

The benefits and limitations of ESSIM's user interface are summarised in figure 71.

BENEFITS

- Ease of use.
- Provision of code generator.
- Mouse support.
- Multiple windowed output.
- Customisable.
- Support for graphics.
- Speeds up development of interface.

LIMITATIONS

- Requires programmer for interfacing work.
- Additional flexibility would be useful.
- Overhead in code size and overhead compared to batch runs.

Figure 71 ESSIM's user interface.

5.10 CONCLUSION

The container port project provided a suitably complex problem to which the ESSIM model development environment could be applied. The three-phase discrete event approach was found to be ideal for the implementation of a model of a container port given the natural breakdown of operations into constituent entities, queues, and activities. The involvement of a company which had an interest in providing suitably experienced managers, enabled realistic scenarios to be enacted through the addition and modification of knowledge-base rules. The details of the port were obtained through a series of informal meetings in which the design principles and expected management procedures were relayed. Subsequent documents were provided which listed expected performance figures. In terms of providing a realistic test-bed for the ESSIM environment, the involvement of a company with a real problem to resolve proved of major benefit.

Because of the number of entities, queues, and activities required in the model, and the overhead resulting from the use of an expert system, a module based version of Pascal was used which enabled the implementation of a program 1Mbyte in size. The use of a modular construct led to the need to modify the three-phase simulation routines and to create a new model template in which life-cycles are represented in separate modules. The use of a modular language also led to the need to introduce new disciplines in code writing. For instance, module interfaces were seen as a necessity in preventing changes introduced in one module from adversely effecting the logic defined in another module. The definition of shared variable also had to be handled with care in preventing illegal use or modification of defined values. The implementation of a simulation environment based on Pascal object modules is considered as being essential to the creation of effective models of real-world problems using PCs. It is unlikely that sufficient detail could be built into a model which is limited by the 64K DOS barrier. In the case of ESSIM in which the Pascal model is supplemented by a knowledge-base interpreted by an expert system inference engine, the use of modules is essential given the amount of code that need to be introduced. Indeed, further detail could not be added to the port model without moving to an alternative operating system on PCs, or by shifting development to a workstation environment.

In section 5.4, the structure and content of the expert system knowledge-base was described. Examples were also given as to the steps required in adding new rules to the system. The 'CraneManager' rule-set was then examined in some detail, tracing the inference process and conclusions drawn by the expert system. The process of developing the knowledge-base for the port model highlighted design errors in the original system. These errors, such as numerical overflows were trapped and corrected by comparing the ESSIM model with the original model in which rules were integrated within the Pascal code. The expert system trace facility was also used in isolating errors by manually scrutinising the logic applied by the inference engine.

Visual inspection of the behaviour of the container port is one of the means available to the modeller in carrying out his analysis. The display is window-based, experience in the manipulation of output having been gained from investigating state-of-the-art techniques in the handling of graphics on PCs. The use of windows and pull-down menus eases the process of manipulating the model and permits the switching of screen output to visually inspect the behaviour of components of each of the port's entity life-cycles. In the past, the benefit of creating windowed user interfaces had to be offset against the overhead in designing screen lay-outs and writing the necessary code. The provision of libraries of standard routines went some way towards simplifying the process, but the need to repeatedly compile and modify a program to achieve the correct relative positioning of screen output still represented a major overhead. With ESSIM, the use of the 'Designer' program generator was found to permit the creation of an effective windowed interface within a time span normally associated with producing simple textual output.

The process of validating the port model was described in section 5.6. Validation is particularly essential in the context of simulation given that a model can behave as would have been expected and yet contain invalid assumptions. The lack of a real-world container port with which model behaviour could be compared complicated the validation process. Given the size of the model, many errors had been expected. In practice, the use of a modular construct combined with the strict rules that had been imposed during the development process had limited the scope for errors. The source of the errors that were made and which were detected from unexpected model behaviour were however often difficult to locate. This is because an error in one module can manifest itself as unexpected behaviour in another module and the process of tracing the source can often be quite complex. In this respect, the trace facilities that were integrated into the port model proved to be of significant use. Validating the expert system knowledge-base was simpler than had been expected. Expert systems use hidden inference strategies which make the normal process of manually tracing through code impossible. Nevertheless, the fact that the rules were isolated from any other code simplified their individual verification on a one-by-one basis. The conclusions drawn by the inference engine, and the results returned to the Pascal model were also easily checked from the output trace which to some extent can be considered as an automation of the manual trace.

In section 5.7, the process of experimenting with the port model was described. The types of experiments that could be conducted were broadly divided into four types: 1) Modification of rule parameter. 2) Altering characteristics of the model such as the defined number of entities and process durations. 3) Altering the structure of existing rule(s) and/or adding further rules associated with existing goals. 4) Adding new goals for which new rules have to be defined. The examples that were given demonstrated that the types of changes listed in 1-3 could, in many cases, be carried with ease and without modification of the Pascal code. The fourth type of change, consisting of adding new goals was far more complex and required a detailed understanding of the underlying simulation code.

In section 5.8, the ESSIM model was compared to two alternative versions of the Port model. The first of these consisted of a straight-forward three-phase model. The second was a version of ESSIM in which the expert system knowledge-base was replaced by a 3GL module in which rule-sets were represented as Pascal functions. The experiments described in section 5.7 were then repeated which demonstrated that the ESSIM model generally offered superior functionality over the alternative approaches.

Finally, section 5.9 reviewed the benefits and limitations of the individual constituents of the ESSIM modelling environment, namely, the three-phase component, the expert system and the graphical user interface.

CHAPTER SIX

CONCLUSION

6.1 REVIEW OF THE THESIS PROPOSITION

The proposition put forward in this thesis was that the use of expert system techniques in the context of simulation, may provide an improved means of structuring the representation of operational policies as enacted by the real-world decision makers. Such operational policies may require the involvement of multiple decision makers, each working at different levels within a hierarchical management structure.

6.2 THE RESEARCH RATIONALE

Discrete event simulation languages predominantly focus on the model representation of physical entities and associated activities. O'Keefe, Belton & Ball[1986] amongst others have noted that there exist limitations in using such languages in representing and subsequently experimenting with the decision tasks normally associated with the management and control of a real-world system.

The use of Artificial Intelligence in the context of simulation modelling has been investigated from a number of angles by several researchers. For instance, attempts have been made at using expert systems during the model building process and for the purpose of interpreting model output. (Such work is of great interest in the context of this thesis and is discussed in chapter two). Flitman & Hurrion [1987] undertook an ambitious project to create a two way link between a simulation model and an AI language. This thesis has followed on from this innovative work by examining the possibility of using an expert system in the representation of complex operational policies and for the use of this representation in subsequent experimentation.

6.3 REVIEW OF THE RESEARCH STRATEGY

The methodology applied to the research described in this thesis has encompassed the following stages:

- A literature review.
- The formalisation of the proposed approach.
- The implementation of a prototype system.
- The development of experimental models using the prototype environment.
- The evaluation of the thesis proposition using the experimental models.

In chapter two, a review was conducted of the two areas of research applicable to this thesis, simulation and artificial intelligence. In the first part of the chapter, simulation modelling and artificial intelligence were examined separately. The second half of the chapter examined the relationship between simulation and AI and subsequently explored the practical work carried out by other researchers.

The literature review was an essential first step in developing an in-depth understanding of the two fields of study and of the work that had already been carried out by other researchers in linking simulation and AI. Having completed the literature study, attention returned to resolving the core issues relating to the thesis proposal. Chapter three consequently began with an analysis of the nature of decision making and sought to address such issues as the representation of hierarchical management structures. Having assessed the problem of representing operational policies within a simulation model, the chapter continued by proposing a potential approach to the implementation of a simulation environment in which the issue of representing operational decision making is specifically addressed.

Chapter four reported the development of a prototype simulation environment (ESSIM) which explores and addresses the technical problems of a linkage between a simulation model and expert system.

The prototype system was used to create a model of an un-manned container port and a number of experiments then carried out based on alternative management rules of varying complexity. Two equivalent models were also developed, one based on a standard simulation framework, and the other seeking to replicate the functionality of the expert system within a purely procedural context. The experiments using alternative management rules which were carried out using the prototype model were repeated using these new models and evidence sought in respect of the suggested potential benefits of the prototype system. The suggested potential benefits identified in the early stages of the thesis can be summarised as follows:

Model Adaptability: It has been proposed that the use of an expert system in the context of simulation modelling, may provide an improved means of representing operational decision making. Furthermore, it has been suggested that the expert system approach could provide the modeller with a simulation environment which is better suited to the task of experimenting with alternative operational policies.

Model accuracy: The ESSIM modelling environment was developed for the purpose of experimenting with operational policies. In doing so, the modeller is expected to define decision making tasks with greater attention to detail than would normally have been expected during a conventional modelling exercise. Consequently one could expect that the modeller's work will result in a model which is a better representation of the real-world problem.

Segregating modelling activities between two modules, one representing the physical system components and the other operational management, was seen as potentially providing further advantages. Such a division could be achieved by retaining a separation between the simulation language and Artificial Intelligence components and implementing a general communications interface between the two. The possible peripheral benefits of a conceptual division were identified to be as follows:

Ease of use: Expert Systems use a high level language syntax akin to fourth generation languages. In addition to this, expert systems use a declarative approach to the definition of rules which removes from the modeller the need to pre-define an execution path. Instead, the expert system uses a generalised inference strategy to control execution. Expert systems have consequently been referred to as being "Easy to use".

Maintainability: The maintainability of a model is related to the ease with which changes that have occurred in the real-world system can be reflected within existing code. The maintainability of the model is of particular importance in situations where changes are expected in the real-world environment during the life of the model. Splitting the model representation between a conventional simulation language and an expert system can be viewed as an extension of the concept of code modularity. The introduction of a further level of modularity may ease the process of making model changes, which in turn could have an impact on the maintainability of the model.

6.4 CONCLUSIONS FROM THE MODEL EXPERIMENTATION

In chapter five, a number of experiments were described which sought to compare the functionality of a model built using ESSIM with that of a conventional Pascal model. The experiments were later repeated using another Pascal model in which an attempt was made at replicating the functionality of the expert system by encapsulating decision rules within Pascal functions. The conclusions from these experiments can be summarised as follows:

Model adaptability: As was reported in section 5.8, it was found that the model written using ESSIM was generally the one which was the easiest to modify for the purpose of evaluating alternative operational policies. The principal reason for this was that decision rules in ESSIM were defined in a far more structured fashion. This structuring of operational decision rules has been referred to in this thesis as "Cooperative decision making". It is assumed in ESSIM that operational policies may be enacted by more than one decision maker. Decision rules are consequently grouped together and identified by the name or function of the decision maker. For the purpose of instigating an operational policy, the rules applied by each of the decision makers may then be brought together. The most senior decision maker may potentially have a veto over any final decision and so his rules would override those of the other decision makers. Such a structural formalism for the representation of operational decision making was difficult to replicate using either of the alternative Pascal models. An example of the type of code which would result from such an attempt is shown in Appendix D.

Model accuracy: As was discussed in section 6.3, it was hoped that the use of ESSIM would result in models which were better representations of the real-world problem. In fact, the issue of model accuracy turns out to be closely related to that of model adaptability. In the previous paragraph, the point was made that the adaptability of the ESSIM model was a consequence of the fact that in ESSIM, decision rules are defined in a far more structured fashion. This structuring of decision rules is what provides an improved representation of the problem being modelled.

The key differences between the ESSIM model of the port and the two equivalent Pascal models are as follows: 1) Decision rules relating to different decision makers are isolated within separate "Rule-sets". 2) Associations may be established between "Rule-sets" in order to reflect the influence of management in the process of decision making. (e.g. A manager may override or simply contribute to the decision taken by a lower ranking employee). 3) Decision rules are defined individually with an inference engine providing the mechanism by which problems are resolved. In the case of the Pascal models, decision rules had to be defined as multi-level conditional statements.

Ease of use: The ease of use of a model is principally related to the language syntax. The expert system used in ESSIM was specifically written for the purpose of simulation modelling. A fourth generation language was used and additional concepts introduced such as "Rule-sets" and "Inheritance" which were aimed at providing a clearer and more structured code lay-out. With respect to the model experiments detailed in Chapter five, such factors gave ESSIM a clear advantage over the equivalent Pascal models in the context of the "Ease of use" of the modelling environment. This advantage was of course limited to model experiments carried out using the expert system knowledge-base. There were a number of other peripheral benefits in using the expert system; 1) The interpreted nature of the system meant that re-compilation could often be avoided. 2) Unlike the Pascal models, decision rules could be defined individually and the problem of resolving a goal left to the inference engine. 3) The isolation of the decision rules into a separate knowledge-base helped in providing a clearer understanding of the function of the rules.

Maintainability: As was discussed in section 6.2, the maintainability of a model is related to the ease with which changes that have occurred in the real-world system can be reflected within existing code. In conventional 3GL programming, the issue of maintainability is in part related to the structure of the code in terms of modularity and syntactic simplicity. It was concluded in section 5.8 that those same issues have an effect on the maintainability of a simulation model. ESSIM extends the concept of code modularity to simulation modelling in a number of respects. 1) In ESSIM, the conventional Pascal model is separated from the decision rules which are located in an expert system knowledge-base. 2) The ESSIM expert system knowledge-base is itself divided into "Rule-sets" which contain the rules applied by individual decision makers. 3) The Pascal simulation code used in ESSIM is divided into separate code modules each containing the code relating to the major entity cycles. The model experiment described in section 5.7.4 required the introduction of a new entity cycle and associated decision rules. Implementing the same changes to each of the three port models highlighted the benefit of a high degree of code modularity. Introducing the model changes was relatively easier to achieve using the ESSIM version. The reason for this is that the use of code modularity in ESSIM introduced a requirement for a higher level of discipline in carrying out the model changes in a step-by-step fashion. In the case of the Pascal models, it was far easier to make a mistake in introducing the model change and subsequently much harder to find where the error had been made.

6.5 SUMMARY OF THE RESEARCH ACHIEVEMENTS

6.5.1 Principal achievements

The research presented in this thesis sought to create a modelling environment in which the principle emphasis is on an adequate representation of operational policies for the purpose of model experimentation. Therefore, the key issues were as follows: Firstly, to identify which aspects of the decision making process had to be represented within the model. Secondly, to identify a modelling approach which was well adapted for the purpose of representing operational decision making problems and which would be appropriate for the purpose of model experimentation. These two steps were successfully completed and as a result, a number of important contributions were made in this field of study.

The key contribution of the research was the introduction of the concept of "Cooperative decision making" which was fully described and discussed in chapters four and five. The expert system knowledge-base is used as a repository for the decision rules applied by selected operational decision makers involved in the day to day running of the real-world system. The research study identified that operational policies often required the cooperation of several decision makers, each controlling a different real-world activity. In other cases, the existence of a hierarchical management structure could mean that a decision taken by one individual could be overridden by another decision maker. Such issues were addressed through the creation of "Rule-sets" and the use of "Inheritance". "Rule-sets" were used for the purpose of segregating the rules applied by each decision maker. The use of "Inheritance", permitted the creation of a logical link between otherwise separate groups of rules.

The introduction of the concept of "Cooperative decision making" brings about a number of fundamental changes to the way in which the modeller builds and subsequently uses the simulation model. Firstly, decision rules are no longer represented as a sequence of conditional statements but require the modeller to represent each operational policy in terms of a goal and related sub-goals. Secondly, the modeller is encouraged to view decision making problems as potentially panning across several activities within a cycle or across multiple activities within <u>different</u> cycles. For instance, an operational policy may require the involvement of more than one decision maker, each represented by a different rule-set and each responsible for the management of entities in different activity cycles.

The testing of the thesis proposition required a comparison to be carried out between the proposed approach and a conventional modelling environment. The research consequently encompassed the development of a prototype simulation environment. This prototype system called ESSIM was successfully developed and consisted of two closely interlinked components, a three-phase discrete event module and expert system. Two code generators were also provided to ease the process of model building. The first of these was used to interactively specify the graphical user interface. The second generator scans the expert system knowledge-base and creates the necessary 3GL code to permit variable sharing between the simulation and expert system modules.

6.5.2 Subsidiary achievements

Other more minor achievements ensued during the course of this thesis. These can be classified into four categories:

6.5.2.1 New approaches to expert system design

As we have seen, the rule-based expert system developed as part of ESSIM, incorporates the concepts of "Rule-sets" and "Inheritance". The use of rule-sets permitted decision rules to be grouped according to the name of the decision maker or by job function. "Inheritance" permitted the inference engine search space to be extended across more than one rule-set. Indirectly, these features helped alleviate two of the classic criticisms of expert systems which are: (1) that the random ordering of rules significantly reduces the legibility and consequent interpretation of the content of the knowledge-base. (2) that expert systems execute slowly. By limiting the backward-chaining search to the rules contained in a specific rule-set, it was possible to substantially improve the speed with which the inference process could be carried out.

The level of integration achieved between a 3GL program and interpreted expert system should be of interest to AI researchers. Again, one of the criticisms of expert systems is that their lack of ability at operating in a procedural context and carrying out complex arithmetical operations prevents their use in a wider context. The ESSIM expert system physically shares the same variable addresses as are defined in the 3GL code, thereby permitting full use of the 3GL variable definitions and providing almost seamless integration.

By being able to trace the addresses of 3GL procedures and functions, the ESSIM expert system is also able to call 3GL code thereby providing the necessary procedural context.

6.5.2.2 Improvements to the three-phase routines

Modifications were made to the library of Pascal simulation routines for three-phase discrete event simulation which help to improve the legibility of the code. For instance, B-event calls are now achieved by placing the start address of the procedure in the executives' timing tree. Hence, at each time advance, B-Events can be activated directly by the executive, without having to first pass control to the 'CallNextBevent' procedure which had to be updated by the modeller to include references to all new B routines.

The application of simulation to the evaluation of alternative decision rules will typically entail the creation of a substantive and complex model. In the case of ESSIM, the development process is further complicated by the use of a third generation language. In order to ease the process of developing large models, the simulation routines used at the LSE were adapted for use in a modular environment based on Object code. Hence, in ESSIM, the modeller is provided with a means of encapsulating each entity cycle within a separate module.

6.5.2.3 Additional software developments

'Designer', the program generator used in creating the graphical user interface was designed using a new approach which again should be of interest to researchers in information technology. The program generator was implemented in such a way that pop-up windows could be created, menus specified and links to external programs defined on an interactive basis. The screen design could then be tested without needing to first generate the underlying code. Most importantly, a generated model can later be modified and subsequently re-linked to the existing program.

6.5.2.4 Peripheral benefits of the ESSIM approach

The linkage of the expert system to a simulation model in conjunction with the implementation of rule-sets has provided a means of 'paralleling' the management structures. In ESSIM multiple decisions may be taken by different individuals at a particular point in time. This ideal is loosely linked to the concept of temporal reasoning and should be of interest to AI researchers.

Another unexpected achievement is the generality of the ESSIM design which could permit the modeller to derive alternative benefits from the linkage of the simulation model and expert system. For instance, one could envisage the simulation model acting under the control of the expert system in order to explore a simulated future as part of a decision making process. The investigation of such possibilities was outside the scope of the thesis but could form the basis of future research work based on ESSIM.

6.6 FUTURE WORK

In it's present state, the ESSIM simulation modelling environment is essentially a prototype system. ESSIM is a complex program which was necessarily developed by a single individual. Limited time was therefore spent on developing such facilities as error handling which would have to be further enhanced if the system was to be used by other researchers. Consequently, it would be desirable as a first step to resolve such problems in order to build a stable system for future research work.

The research presented in this thesis has highlighted the considerable potential that simulation modelling offers in the context of the evaluation of alternative operational policies. It is the author's opinion that there are yet major advances to be made in this area which could result in a far greater acceptance of simulation modelling as a management tool. Of all the possible research projects that could be established based on ESSIM, the greatest potential possibly lies in developing a system like ESSIM along the lines of a management decision support system. The work presented in this thesis has already served to demonstrate that by isolating the model representation of the physical components of a real-world system from operational decision rules, model experiments could be carried out by solely altering defined decision rules. Two essential steps would be required in bringing this kind of modelling into the realms of direct end-user decision support. Firstly, further advances would be required in the development of tools for the automated interpretation of model output. Secondly, the creation and subsequent modification of decision rules would need to be further simplified, possibly through the introduction of menuing systems or natural language interfaces.

APPENDIX A

CONTENTS - APPENDIX A

APPENDIX A

JOB-SHOP PRODUCTION SCHEDULING USING ESSIM

A.1 INTRODUCTION

Mathematical modelling has been used in the context of the job-shop, in evaluating proposed heuristics and in generating alternatives which may be shown to be 'optimal' in some sense. Such mathematical modelling has inevitably been based on rather simple descriptions of the job shop environment. Consequently, discrete event simulation modelling has often been used for rule evaluation in more realistic contexts (Arumugam[1985], Barrett & Barman[1986] and Kiran & Smith[1983]).

Simulation models have been used extensively in job-shop production environments in evaluating different scheduling rules, in developing scheduling rules under a given set of parameters, and in analysing the sensitivity of job-shop models to changes in scheduling rules. Such simulations require access to relevant databases and must reflect the complex decision-making of the job shop. If this is to be achieved without demanding extensive intervention from the production controller then the decision-making capacity must be accommodated within the simulation model.

A production controller in a typical engineering job shop relies in great measure on his own personal experience and judgement in scheduling workloads. These decisions cannot be made without a detailed knowledge of the current state of material stocks, outstanding orders and work-in-progress. On-line decision support systems for job shop scheduling have been introduced to provide such necessary information through the implementation or interfacing of appropriate database systems. More recently, the provision of information has been supplemented by the addition of rule-based expert systems which seek to encapsulate explicitly, in a collection of scheduling rules or heuristics, the experience and knowledge of the production controller (Fox and Smith[1984]). The rule-base appropriate to a particular job shop context will typically be complex and requires painstaking development in collaboration with production management.

Consequently, the potential synergy of simulation and expert system would seem a logical step forward in the context of the job shop. ESSIM, in this respect appears to be an appropriate development tool. The remaing sections describe the development of a job-shop model initially based on experiences gained from two manufacturing concerns operating from Rio de Janeiro, Brasil (Costa and Jardim[1986]).

A.2 OVERALL SYSTEM DESIGN

Figure A72 shows the overall structure of the system. The simulation model progresses the job-shop operations through time. The expert system manages the scheduling of tasks within the simulated system. User defined databases are used for the storage of data relating to product descriptions and outstanding orders. The user interface system is used for the control of output displays.

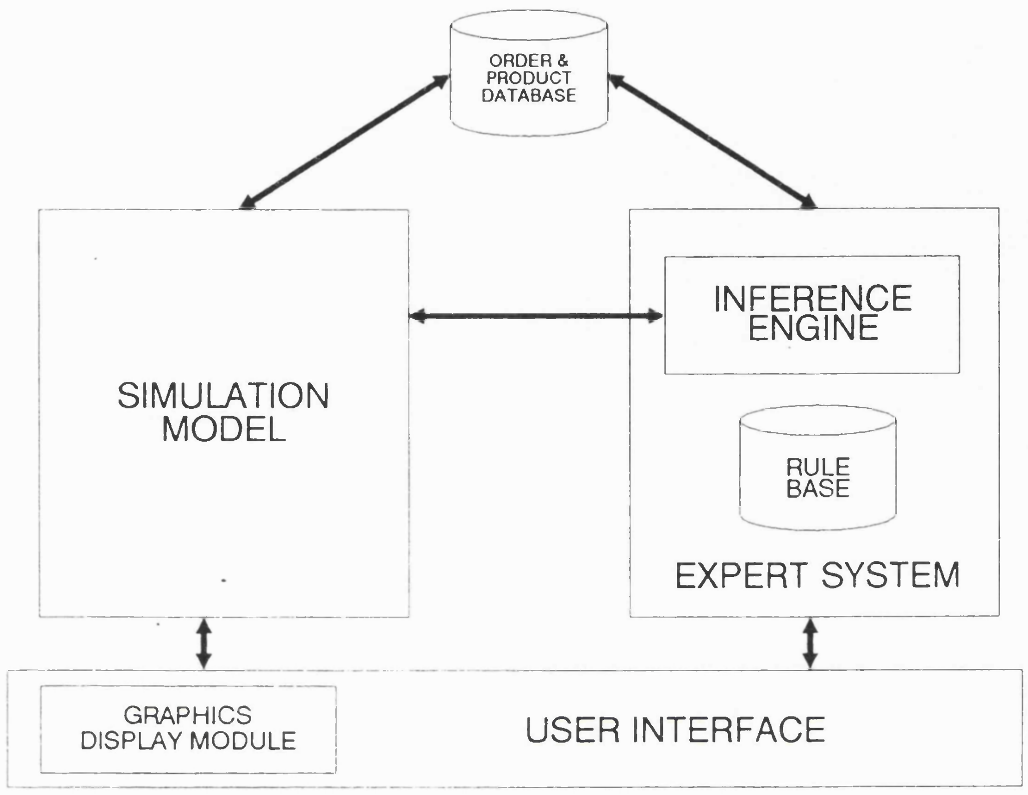


Figure A72 System overview for the job-shop model

The description of the job shop is in terms of a number of 'work centres'. Each work centre is comprised of a number of collaborative facilities, such as machines and associated operatives, which regularly work together as a unit in the performance of individual production operations. A work centre is characterised by the range of operations of which it is capable and parameters, such as cost, quality or speed, of its performance of these.

A 'product' may be characterised by its material components and the possible sequences of operations, such as machining, assembly, painting etc., and the associated work centres on which these operations might be performed.

The technical analysis of a given physical product into the characterising sequences of production possibilities is a task left by the present system to the production engineer. Product descriptions are maintained in the 'Product Database' of Figure A72 and referenced by the expert system in determining scheduling possibilities. Supply of materials from inventory is considered as the function of a particular work centre which must also manage an inventory control policy.

An 'order' is characterised by the identity of the customer, the product and quantity requested, the time of placement, due date and priority rating accorded by sales staff. A file of orders received, in-progress and completed is maintained as a specification of the work requirement of the shop and a measure of performance.

A.3 THE SIMULATION MODEL

The job-shop model uses ESSIMs simulation module which is structured according to the Three Phase system (see section 2.4.1). The A phase searches a diary of prescheduled events, such as the completion of an operation by a work centre, for the earliest scheduled event and advances the model clock to this time. The B phase manages the execution of all events scheduled to occur at the current clock time and the C phase explores the resulting model state to determine what new events might be scheduled. This latter phase is typically where the decision is made to start work on a particular task and hence enter into the diary the scheduled time of its completion.

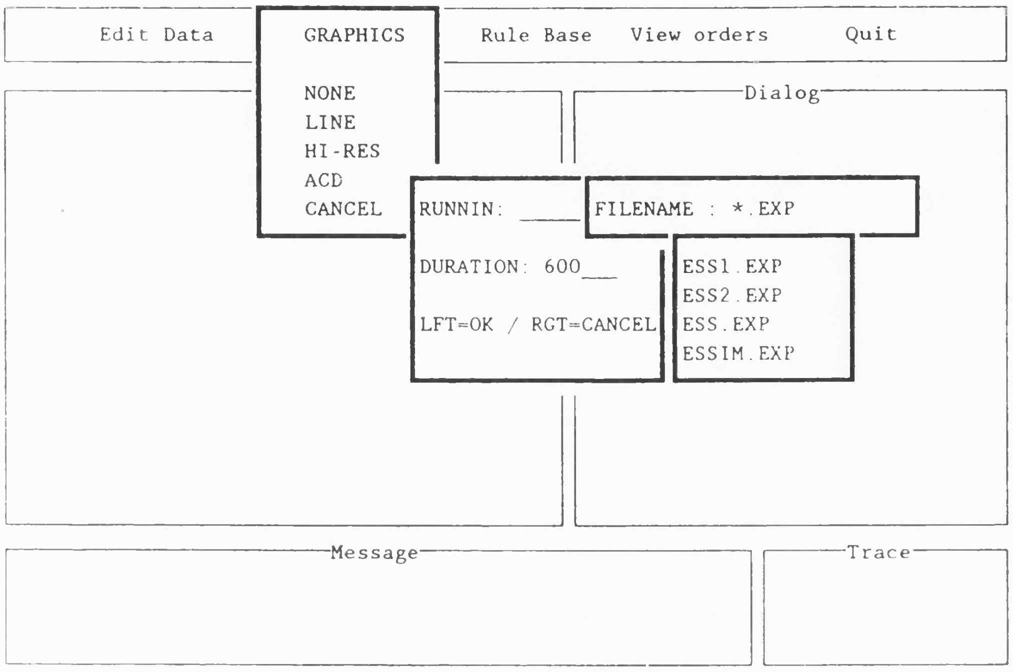
The principal system entities are the work centres which follow a simple life cycle alternating between periods of processing jobs and idleness. Each work centre has a list of allocated tasks, some of which may be suspended pending the arrival of necessary components, from which a next task may be selected.

The arrival of orders and consequent inflow of further jobs may either be on the specific intervention of the user, from a predefined file of orders or according to a randomised arrival mechanism. On receipt each order is translated into its components and incorporated into the job shop schedule.

A.4 THE USER INTERFACE

The job-shop model makes use of an interface which is largely menu-driven from a multi-window screen. The interface design was manually created rather than using the ESSIM interface generator, and so output is largely text based. The ESSIM interface design was developed following from the experiences gained from implementing the job-shop model interface.

The basic screen is shown in Figure A73. The various windows contain the main menu offering options, 'Simulate', 'Edit Product File', 'Edit Rule Base' and 'View Orders'. Subsidiary pull-down menus allow a further range of options. Selection of options is effected under cursor control, by initial letter or using a mouse. Figure A73 actually shows some of the sub-menus which are presented to the user before the simulation begins.



RUNNIN: ____

FILENAME : *.EXP

DURATION: 600___

FIGURE A73 THE JOB-SHOP APPLICATION INTERFACE

The windows provide further areas for system 'Messages', textual 'Output' during the execution of the simulation, user-machine 'Dialog' and execution 'Trace'. Simple, directed editors permit the examination, addition and deletion of entries in product and order files and rule-base.

The dynamic behaviour of the simulated job shop may be studied through alternative display formats. The user may select from a textual display, a diagrammatic representation of the job shop through the life cycle of the work centres or through high resolution graphs of the accumulating work loads of a selected set of work centres.(see Figures A74 and A75)

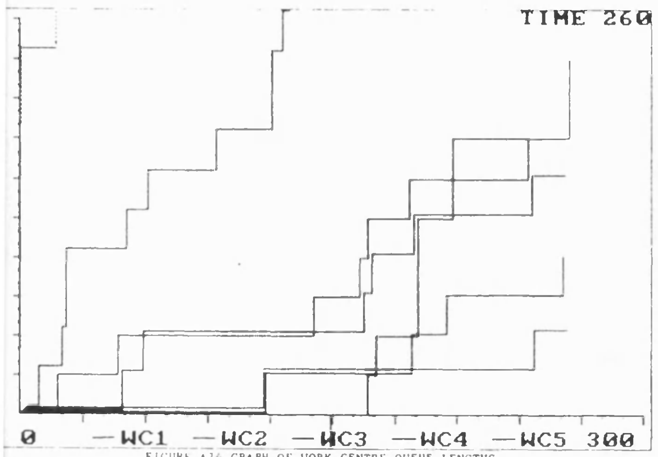


FIGURE A74 GRAPH OF WORK-CENTRE QUEUE LENGTHS

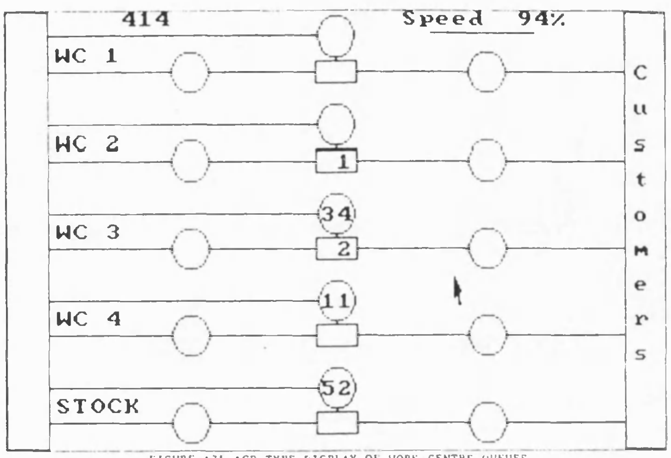


FIGURE A75 ACD TYPE DISPLAY OF WORK CENTRE QUEUES

Details of the orders completed and those still outstanding can be viewed on request either during or following the execution of the simulation program.

A.5 THE EXPERT SYSTEM AND INTERFACES WITH THE SIMULATION MODEL

During the execution of the simulation model, the C-phase of every cycle requires that decisions be made as to which activity, if any should start at that time. When a work centre finishes its current task, it must select the next job from those possible. When a new order arrives, it must either be rejected or the component tasks must be allocated amongst the appropriate work centres. The simulation executive looks to the associated expert system to specify such selections and allocations. The current state of the simulation model provides the context within which the expert system applies its knowledge and thus the expert system is able to access the simulation state variables.

The knowledge base of the expert system is defined in terms of a structured file of IF-THEN production rules. The structure reflects the intrinsic hierarchical structure of production planning processes and is described in detail in section 3.2 (see figure 8 and Bitran et al.[1982], Erschler et al.[1986], and Bullers & Schultz[1986]).

The decision as to which job to start at a given work centre may be made with reference to the local environment of that work centre. i.e. the job may be selected from the list of possible jobs currently available according to local operating constraints and sequencing rules. These may include specific management direction as to which rules to apply at particular times or with respect to particular jobs.

The job-shop application was developed as a means of gaining understanding of the requirement of the simulation modeller. A lack of detailed rules relating to the operation of the production plant made the implementation of a realistic knowledge-base difficult. The job-shop application was consequently used for the validation of the expert system and its links to the simulation model. This formed the basis for the development of the ESSIM environment and its subsequent validation using the port model.

APPENDIX B

APPENDIX B

THE PORT MODEL KNOWLEDGE-BASE UNDER ESSIM

```
EXTERNAL {declaration of integer variables to be shared with the simulation
        model}

    NumbStackRmgIdle, NumbMoveToShipJob, NumbMoveToExitJob,
    NumbRmgAtSSide, NumbRmgAtLSide, RmgToLeaveSSide,
    RmgToLeaveLSide, NumbTruckToBay, BayPos,
    NumTruckOutside,ShipImportJobs,ShipExportJobs,NumShipCranesIdle,
    NumTotalImportJobs, NumCranesOnImports, CurrentShipBerth,
    NumberOfGateVehicles,NumberOfJobs,NumberOfBerths,NumberOfImvs,
    NumberOfShipCranes, NumbFullImvsToStore, _Time,
    NumbEmptyImvsToStore,ExportsLeftForShip,StoreNumWithExpContainer,
    NumberOfShips, NumberOfStores, NumbEmptyImvToShip,
    NumbFullImvToShip, NumberOfShipsAtSea, NumEmptyImvAtShip,
    NumEmptyImvAtBerth, NumEmptyImvAtOtherBerth,
    NumIdleImvs, NumFullImvAtBerth, NumImportsRemaining,
    NumbShipsWaitingToBerth, NumbFreeBerths, TimeToUnloadFullImv,
    TimeToFetchImvFromShipQ, TimeToUnLoadShip, TimeToLoadShip,
    TimToGetImvFromOtherShipQ, TimeToFetchImvFromIdleQ,
    TimeToMoveToShipEmpty, TimeToMoveToShipFull,
    TimeToMoveToLandEmpty, TimeToMoveToLandFull,
    TimeToGetAuthorization : INTEGER ;
    {declaration of boolean variables to be shared with the simulation model}
    CraneOperational, ShipInBerthOne, ShipInBerthTwo, CraneOnImports,
    CraneLoaded, _LoadImv, _UnLoadImv, _LoadShip, _UnLoadShip,
    GetAuthorization, AuthorizeCraneToExport, ReturnImvsToIdle,
    FetchImvFromShipQ, FetchImvFromOtherShipQ, FetchImvFromIdleQ,
    _FullImvToUnload, _ChangeACraneToExports, _MakeImvIdle,
    ShipArrivalDue, WorkAtBerthCompleted, _MoveGantryToShipSide,
    _MoveGantryToLandSide, MoveGantryToShipEmpty,
    MoveGantryToLandEmpty, NextShipJobIsAnImport,
    NextLandJobIsAnImport, LoadVehicleWithImport,
    UnLoadExportFromVehicle, NextJobIsAnImport, MoveImportToStack,
    MoveEmptyToStack, MoveEmptyToStackFromGate, MoveExportToStack,
    LoadImportFromImv, UnloadExportToImv: BOOLEAN ;

PASCAL FILE  'Rules.pas' ;   {File of Pascal routines to be called from within
                              the expert system.}

LOCAL {declaration of local expert system boolean variables}
    StartlandSideWork, StartShoreSideWork, StartNewJob,
    StartMoveFromSSide, StartMoveFromLSide, MoveToBay, BayFree,
    TruckWaiting, TruckOutside, WaitingForTruck, ShipJobsLeft,
    LetTruckInPort, BerthedShip, CraneJobs, NoExportsInStore,
    SendFullImvToStore, SendEmptyImvToShip, SendFullImvToShip,
    StartShipArrive, StartShipLeave, StartDockAtBerth, ShipFullOfExports,
    ImvsIdle, SendEmptyImvToStore, EmptyImvToLoad, JobToBeDone,
    JobToDo, PriorityToShipjobs, JobOutstanding, NextJob,
    JobFound, CraneInCorrectMode, ShipToLoad, ShipToUnload : Boolean;
```

```
      {Declaration of local expert system intenger variables}

      Bay : INTEGER ;



RULESET StackManager ;


[*] NumberOfStores = 10;       {number of stores is 10}


[*] NumberOfJobs = 50 ;        {number of jobs is 50}


[1] STARTNEWJOB = True WHEN (NumbStackRmgIdle > 0)
                  AND (JobToBeDone = True) ;


[2] PriorityToShipJobs = GantryToShipSide ; {enquiry to rules.pas}


[3] _MoveGantryToShipSide = True
                  IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0)
                  AND (PriorityToShipJobs = True)) ;


[4] _MoveGantryToLandSide = True
                  IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0)
                  AND (PriorityToShipJobs = False)) ;


[5] _MoveGantryToShipSide = True WHEN NumbMoveToShipJob > 0 ;


[6] _MoveGantryToLandSide = True WHEN NumbMoveToExitJob > 0 ;
```

```
[7] (MoveGantryToShipEmpty = True) AND (JobToBeDone = True)
                  AND (_MoveGantryToLandSide = False)
                  AND (TimeToMoveToShipEmpty = 40)
                  IF (_MoveGantryToShipSide = True)
                  AND (NextShipjobIsAnImport = True) ;

[8] (MoveGantryToShipEmpty = False) AND (JobToBeDone = True)
                  AND (_MoveGantryToLandSide = False)
                  AND (TimeToMoveToShipFull = 60)
                  IF (_MoveGantryToShipSide = True)
                  AND (NextShipjobIsAnImport = False);

[9] (MoveGantryToLandEmpty = True) AND (JobToBeDone = True)
                  AND (_MoveGantryToShipSide = False)
                  AND (TimeToMoveToLandEmpty = 40)
                  IF (_MoveGantryToLandSide = True)
                  AND (NextLandJobIsAnImport = False) ;

[10] (MoveGantryToLandEmpty = False) AND (JobToBeDone = True)
                  AND (_MoveGantryToShipSide = False)
                  AND (TimeToMoveToLandFull = 60) ~ (JobToBeDone = False)
                  IF (_MoveGantryToLandSide = True)
                  AND (NextLandJobIsAnImport = True) ;
```

```
[11] (JobOutstanding = True) AND (LoadImportFromImv = True)
                 AND (UnloadExportToImv = False) AND (_Time = 60)
                 ˜ (JobOutstanding = True) AND (UnloadExportToImv = True)
                 AND (_Time = 60) AND (LoadImportFromImv = False)
                 IF NextJobIsAnImport = True;

[12] STARTSHORESIDEWORK = True WHEN (NumbRmgAtSSide > 0)
                 AND (JobOutstanding = True);

[13] (JobToDo = True) AND (LoadVehicleWithImport = True)
                 AND (UnloadExportFromVehicle = False) AND (_Time = 60)
                 ˜ (JobToDo = True) AND (UnloadExportFromVehicle = True)
                 AND (_Time = 60) AND (LoadVehicleWithImport = False)
                 IF NextJobIsAnImport = True;

[14] STARTLANDSIDEWORK = True WHEN (NumbRmgAtLSide > 0)
                 AND (JobToDo = True);

[15] (NextJob = True) AND (MoveImportToStack = True)
                 AND (MoveEmptyToStack = False) AND (_Time = 60)
                 ˜ (NextJob = True) AND (MoveEmptyToStack = True)
                 AND (_Time = 40) AND (MoveImportToStack = False)
                 IF NextJobIsAnImport = True;

[16] STARTMOVEFROMSSIDE = True WHEN (RmgToLeaveSSide > 0)
                 AND (NextJob = True);
```

```
[17] (JobFound = True) AND (MoveEmptyToStackFromGate = True)
                 AND (MoveExportToStack = False) AND (_Time = 40)
                 ~ (JobFound = True)
                 AND (MoveEmptyToStackFromGate = False) AND (_Time = 60)
                 AND (MoveExportToStack = True)
                 IF NextJobIsAnImport = True;


[18] STARTMOVEFROMLSIDE = True WHEN (RmgToLeaveLSide > 0)
                 AND (JobFound = True);


[19] Bay = WhichBay ;


[20] (bayFree = True) AND (BayPos = Bay) ~ (bayFree = False) IF Bay > 0 ;


[21] NoExportsInStore = True WHEN StoreNumWithExpContainer = 0 ;



RULESET GateManager (INHERIT StackManager,Shipmanager) ;


[*] NumberOfGateVehicles = 50 ; {Number of gate vehicles is 50}


[1] (TruckOutSide = True) ~ (TruckOutside = False)
                 AND (waitingForTruck = False)
                 IF NumTruckOutside > 0 ;


[2] waitingForTruck = True WHEN shipJobsLeft = True;
```

```
[3] (truckWaiting = True) ~ (truckWaiting = False) AND (bayFree = False)
                    IF NumbTruckToBay > 0 ;

[4] LETTRUCKINPORT = True WHEN (truckOutside = True)
                    AND (waitingForTruck = True);

[5] (MOVETOBAY = True) AND (_Time = 0) ~ (MOVETOBAY = FALSE)
                    IF (truckWaiting = True) and (bayFree = True) ;


RULESET ShipManager ;

[*] NumberOfShips = 9 ; {maximum number of ships is 9}

[*] NumberOfBerths = 2 ; {maximum number of berths is 2}

[1] shipJobsLeft = True WHEN (ShipImportJobs > 0)
                    OR (ShipExportJobs > 0) ;

[2] berthedShip = True WHEN (ShipInBerthOne = True)
                    OR (ShipInBerthTwo = True);

[3] shipFullOfExports = True WHEN ExportsLeftForShip = 0 ;
```

```
[4] (STARTSHIPARRIVE = True) AND (_Time = 1200)
                    ~ (STARTSHIPARRIVE = False)
                    IF (NumberOfShipsAtSea > 0)
                    AND (ShipArrivalDue = True) ;

[5] (STARTDOCKATBERTH = True) AND (_Time = 120)
                    ~ (STARTDOCKATBERTH = False)
                    IF (NumbShipsWaitingToBerth > 0)
                    AND (NumbFreeBerths > 0) ;

[6] (STARTSHIPLEAVE = True) AND (_Time = 3600)
                    ~ (STARTSHIPLEAVE = False)
                    IF WorkAtBerthCompleted = True ;

[7] ShipToload = True WHEN ((CurrentShipBerth = 1)
                    AND (ShipInBerthOne = True))
                    OR ((CurrentShipBerth = 2) AND (ShipInBerthTwo = True));

[8] ShipToUnload = True WHEN ((CurrentShipBerth = 1)
                    AND (ShipInBerthOne = True))
                    OR ((CurrentShipBerth = 2) AND (ShipInBerthTwo = True));


RULESET ImvManager (INHERIT ShipManager,StackManager);

[*] NumberOfImvs = 100 ; {maximum number of imvs is 100}
```

```
[1] (imvsIdle = True) AND (ReturnImvsToIdle = True) ~ (imvsIdle = False)
                    IF (berthedShip = False) AND (NumEmptyImvAtShip > 0);

[2] (emptyImvToLoad = True) AND (FetchImvFromShipQ = True)
                    AND (FetchImvFromOtherShipQ = False)
                    AND (FetchImvFromidleQ = False)
                    AND (TimeToFetchImvFromShipQ = 40)
                    IF NumEmptyImvAtBerth > 0 ;

[3] (emptyImvToLoad = True) AND (FetchImvFromOtherShipQ = True)
                    AND (FetchImvFromShipQ = False)
                    AND (FetchImvFromIdleQ = False)
                    AND (TimToGetImvFromOtherShipQ = 40)
                    IF NumEmptyImvAtOtherBerth > 0 ;

[4] (emptyImvToLoad = True) AND (FetchImvFromIdleQ = True)
                    AND (FetchImvFromShipQ = False)
                    AND (FetchImvFromOtherShipQ = False)
                    AND (TimeToFetchImvFromIdleQ = 40)
                    ~ (emptyImvToLoad = False) IF NumIdleImvs > 0 ;

[5] (_FullImvToUnLoad = True) AND (TimeToUnloadFullImv = 40)
                    ~ (_FullimvToUnload = False) IF NumFullImvAtBerth > 0 ;
```

```
[6] (SENDEMPTYIMVTOSTORE = True) AND (_Time = 30)
                  ~ (SENDEMPTYIMVTOSTORE = False)
                  IF (NumbEmptyImvsToStore > 0)
                  AND (_MakeImvIdle = False) ;

{SENDEMPTYIMVTOSTORE Must return _MakeImvIdle}

[7] _MakeImvIdle = True WHEN (shipFullOfExports = True)
                  OR (noExportsInStore = True) ;

[8] (SENDFULLIMVTOSTORE = True) AND (_Time = 60)
                  ~ (SENDFULLIMVTOSTORE = False)
                  IF (NumbFullImvsToStore > 0);

[9] (SENDEMPTYIMVTOSHIP = True) AND (_Time = 60)
                  ~ (SENDEMPTYIMVTOSHIP = False)
                  IF (NumbEmptyImvToShip > 0) ;

[10] (SENDFULLIMVTOSHIP = True) AND (_Time =120)
                  ~ (SENDFULLIMVTOSHIP = False)
                  IF (NumbFullImvToShip > 0) ;
```

```
RULESET CraneManager (INHERIT ImvManager,ShipManager);

{CRANEJOBS must return _LoadImv, _UnLoadImv, _LoadShip, _UnloadShip,
     FetchImvFromShipQ, FetchImvFromOtherShipQ, FetchImvFromIdleQ,
     _FullImvToUnload, ReturnImvsToIdle, _ChangeACraneToExports,
    GetAuthorization}

[*] NumberOfShipCranes = 5 ;  {Total No. of ship cranes}

[*] TimeToLoadShip = 40 ;

[*] TimetoUnloadShip = TimeToUnloadCalc ; {Call to Pascal function}

[1] CRANEJOBS = False IF CraneOperational = False ;

[2] CRANEJOBS = True WHEN ((_LoadImv = True) OR (_UnLoadImv = True)
                  OR (_LoadShip = True) OR (_UnLoadShip = True))
                  AND (CraneInCorrectMode = True) ;

[3] CraneInCorrectMode = True WHEN (CraneOperational = True)
                  AND ((CraneOnImports = True) AND ((_LoadImv = True)
                  OR (_UnloadShip = True))) OR ((CraneOnImports = False)
                  AND ((_UnLoadImv = True) OR (_LoadShip = True)));

[4] _LoadImv = True WHEN (CraneLoaded = True)
                  AND (emptyImvToLoad = True) ;
```

```
[5] _UnLoadImv = True WHEN (CraneLoaded = False)
                 AND (_FullImvToUnLoad = True);

[6] _LoadShip = True WHEN (CraneLoaded = True) AND (ShipToLoad = True);

[7] _UnLoadShip = True WHEN (_ChangeACraneToExports = False)
                 AND (CraneLoaded = False) AND (ShipToUnload = True) ;

[8] (_ChangeACraneToExports = True) AND (GetAuthorization = True)
                 AND (TimeToGetAuthorization = CalcFromCurrentTime)
                 {Call to Pascal function}
                 AND (CraneOperational = False)
                 ~ (_ChangeACraneToExports = False)
                 AND (GetAuthorization = False)
                 IF ((CurrentShipBerth = 1)
                 AND (NumCranesOnImports > 0))
                 OR ((CurrentShipBerth = 2)
                 AND (NumCranesOnImports > 1))
                 AND (NumImportsRemaining = 0)
                 AND (NumTotalImportJobs > 0);
```

```
[9] (AUTHORIZECRANETOEXPORT = True) AND (CraneOperational = True)
                AND (CraneOnImports = False)
                ~ (AuthorizeCraneToExport = False)
                IF ((CurrentShipBerth = 1)
                AND (NumCranesOnImports > 0))
                OR ((CurrentShipBerth = 2)
                AND (NumCranesOnImports > 1))
                AND (NumImportsRemaining = 0)
                AND (NumTotalImportJobs > 0);

END.
```

APPENDIX C

APPENDIX C

ADDITION OF A RULE-SET TO THE KNOWLEDGE-BASE

```
RULESET StackManager (INHERIT LMVmanager) ;

[*] NumberOfStores = 10;      {maximum number of stores is 10}

[*] NumberOfJobs = 50 ;       {maximum number of jobs is 50}

[1] STARTNEWJOB = True WHEN (NumbStackRmgIdle > 0)
                  AND (JobToBeDone = True) ;

[2] PriorityToShipJobs = GantryToShipSide ; {enquiry to rules.pas}

[3] _MoveGantryToShipSide = True
                  IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0)
                  AND (PriorityToShipJobs = True)) ;

[4] _MoveGantryToLandSide = True
                  IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0)
                  AND (PriorityToShipJobs = False)) ;

[5] _MoveGantryToShipSide = True WHEN NumbMoveToShipJob > 0 ;

[6] _MoveGantryToLandSide = True WHEN NumbMoveToExitJob > 0 ;
```

```
[7] (MoveGantryToShipEmpty = True) AND (JobToBeDone = True)
                   AND (_MoveGantryToLandSide = False)
                   AND (TimeToMoveToShipEmpty = 40)
                   IF (_MoveGantryToShipSide = True)
                   AND (NextShipjobIsAnImport = True) ;

[8] (MoveGantryToShipEmpty = False) AND (JobToBeDone = True)
                   AND (_MoveGantryToLandSide = False)
                   AND (TimeToMoveToShipFull = 60)
                   IF (_MoveGantryToShipSide = True)
                   AND (NextShipjobIsAnImport = False);

[9] (MoveGantryToLandEmpty = True) AND (JobToBeDone = True)
                   AND (_MoveGantryToShipSide = False)
                   AND (TimeToMoveToLandEmpty = 40)
                   IF (_MoveGantryToLandSide = True)
                   AND (NextLandJobIsAnImport = False) ;

[10] (MoveGantryToLandEmpty = False) AND (JobToBeDone = True)
                   AND (_MoveGantryToShipSide = False)
                   AND (TimeToMoveToLandFull = 60)
                   ~ (JobToBeDone = False) IF (_MoveGantryToLandSide = True)
                   AND (NextLandJobIsAnImport = True) ;
```

```
[11] (JobOutstanding = True) AND (LoadImportFromImv = True)
                 AND (UnloadExportToImv = False) AND (_Time = 60)
                 ~ (JobOutstanding = True) AND (UnloadExportToImv = True)
                 AND (_Time = 60) AND (LoadImportFromImv = False)
                 IF NextJobIsAnImport = True;

[12] STARTSHORESIDEWORK = True WHEN (NumbRmgAtSSide > 0)
                 AND (JobOutstanding = True);

[13] (JobToDo = True) AND (LoadVehicleWithImport = True)
                 AND (UnloadExportFromVehicle = False) AND (_Time = 60)
                 ~ (JobToDo = True) AND (UnloadExportFromVehicle = True)
                 AND (_Time = 60) AND (LoadVehicleWithImport = False)
                 IF NextJobIsAnImport = True;

[14] STARTLANDSIDEWORK = True WHEN (NumbRmgAtLSide > 0)
                 AND (JobToDo = True);

[15] (NextJob = True) AND (MoveImportToStack = True)
                 AND (MoveEmptyToStack = False) AND (_Time = 60)
                 ~ (NextJob = True) AND (MoveEmptyToStack = True)
                 AND (_Time = 40) AND (MoveImportToStack = False)
                 IF NextJobIsAnImport = True;

[16] STARTMOVEFROMSSIDE = True WHEN (RmgToLeaveSSide > 0)
                 AND (NextJob = True);
```

```
[17] (JobFound = True) AND (MoveEmptyToStackFromGate = True)
                AND (MoveExportToStack = False) AND (_Time = 40)
                ~ (JobFound = True)
                AND (MoveEmptyToStackFromGate = False) AND (_Time = 60)
                AND (MoveExportToStack = True)
                IF NextJobIsAnImport = True;

[18] STARTMOVEFROMLSIDE = True WHEN (RmgToLeaveLSide > 0)
                AND (JobFound = True);

[19] Bay = WhichBay ;

[20] (bayFree = True) AND (BayPos = Bay) ~ (bayFree = False) IF Bay > 0 ;

[21]  NoExportsInStore = True WHEN StoreNumWithExpContainer = 0 ;

[22] (BALANCESTACKS = True) AND (_Time = 60)
                ~ (BALANCESTACKS = FALSE)
                IF (_TransfExpToRhtStack = True)
                OR (_TransfExpToLftStack = True)
                OR (_TransfImpToRhtStack = True)
                OR (_TransfImpToLftStack = True)
                AND (IdleStackVehicles = True) ;
```

```
RULESET LMVmanager ;

[1]  MoreExptsInLeftStack = True
                    WHEN (ExportsInStackLeft - ExportsInStackRight) > 0 ;

[2]  _TransfExpToRhtStack = True WHEN (MoreExptsInLeftStack = True)
                    AND (ExportsInStackRight < ExportsInCurrentStack) ;

[3]  _TransfExpToLftStack = True WHEN (MoreExptsInLeftStack = False)
                    AND (ExportsInStackLeft < ExportsInCurrentStack) ;

[4]  MoreImptsInLeftStack = True
                    WHEN (ImportsInStackLeft - ImportsInStackRight) > 0 ;

[5]  _TransfImpToRhtStack = True WHEN (MoreImptsInLeftStack = True)
                    AND (ImportsInStackRight < ImportsInCurrentStack) ;

[6]  _TransfImpToLftStack = True WHEN (MoreImptsInLeftStack = False)
                    AND (ImportsInStackLeft < ImportsInCurrentStack) ;
```

APPENDIX D

APPENDIX D

CODING THE PORT MODEL KNOWLEDGE-BASE IN PASCAL

```
MODULE expertrules ;

VAR[External] {Variables shared with the simulation module}

        NumbStackRmgIdle, NumbMoveToShipJob, NumbMoveToExitJob,
        NumbRmgAtSSide, NumbRmgAtLSide, RmgToLeaveSSide,
        RmgToLeaveLSide, NumbTruckToBay, BayPos, NumTruckOutside,
        ShipImportJobs, ShipExportJobs, NumShipCranesIdle,
        NumTotalImportJobs, NumCranesOnImports, CurrentShipBerth,
        NumberOfGateVehicles, NumberOfJobs, NumberOfBerths, NumberOfImvs,
        NumberOfShipCranes, NumbFullImvsToStore, _Time,
        NumbEmptyImvsToStore,ExportsLeftForShip,StoreNumWithExpContainer,
        NumberOfShips, NumberOfStores, NumbEmptyImvToShip,
        NumbFullImvToShip, NumberOfShipsAtSea, NumEmptyImvAtShip,
        NumEmptyImvAtBerth, NumEmptyImvAtOtherBerth, NumIdleImvs,
        NumFullImvAtBerth, NumImportsRemaining, NumbShipsWaitingToBerth,
        NumbFreeBerths, TimeToUnloadFullImv, TimeToFetchImvFromShipQ,
        TimeToUnLoadShip, TimeToLoadShip, TimToGetImvFromOtherShipQ,
        TimeToFetchImvFromIdleQ, TimeToMoveToShipEmpty,
        TimeToMoveToShipFull, TimeToMoveToLandEmpty,
        TimeToMoveToLandFull : INTEGER ;

        CraneOperational, ShipInBerthOne, ShipInBerthTwo, CraneOnImports,
        CraneLoaded, _LoadImv, _UnLoadImv, _LoadShip, _UnLoadShip,
      ReturnImvsToIdle, FetchImvFromShipQ, FetchImvFromOtherShipQ,
      FetchImvFromIdleQ, _FullImvToUnload, ChangeACraneToExports,
      _MakeImvIdle, ShipArrivalDue, WorkAtBerthCompleted,
      _MoveGantryToShipSide , _MoveGantryToLandSide,
      MoveGantryToShipEmpty, MoveGantryToLandEmpty,
      NextShipJobIsAnImport,NextLandJobIsAnImport,LoadVehicleWithImport,
      UnLoadExportFromVehicle, NextJobIsAnImport, MoveImportToStack,
      MoveEmptyToStack, MoveEmptyToStackFromGate, MoveExportToStack,
      LoadImportFromImv, UnloadExportToImv: BOOLEAN ;

VAR  {variables local to this module}
      BayFree, TruckWaiting, TruckOutside, WaitingForTruck, ShipJobsLeft,
      BerthedShip, NoExportsInStore, ShipFullOfExports, ImvsIdle,
      EmptyImvToLoad, JobToBeDone, PriorityToShipjobs, JobOutstanding,
      NextJob, JobFound:Boolean;

      Bay : INTEGER ;


FUNCTION GantryToShipSide : Boolean; Extern;


FUNCTION WhichBay: Integer ; Extern;
FUNCTION StartNewJob : Boolean ;
BEGIN
PriorityToShipJobs := GantryToShipSide ; {enquiry to rules.pas}
IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0) AND
      (PriorityToShipJobs = True)) THEN
      BEGIN
            _MoveGantryToShipSide := True;
            _MoveGantryToLandSide := False ;
      END ELSE
IF ((NumbMoveToShipJob * NumbMoveToExitJob > 0) AND
      (PriorityToShipJobs = False)) THEN
      BEGIN
            _MoveGantryToLandSide := True;
            _MoveGantryToShipSide := False ;
      END ELSE
IF NumbMoveToShipJob > 0 THEN
      BEGIN
            _MoveGantryToShipSide := True ;
            _MoveGantryToLandSide := False;
      END ELSE
IF NumbMoveToExitJob > 0 THEN
      BEGIN
            _MoveGantryToLandSide := True ;
            _MoveGantryToShipSide := False ;
      END ELSE
      BEGIN
            _MoveGantryToLandSide := False ;
            _MoveGantryToShipSide := False ;
      END;
IF (_MoveGantryToShipSide = True) AND (NextShipjobIsAnImport = True)
THEN
      BEGIN
            MoveGantryToShipEmpty := True;
            JobToBeDone := True;
            _MoveGantryToLandSide := False;
            TimeToMoveToShipEmpty := 40;
      END ELSE
IF (_MoveGantryToShipSide = True) AND (NextShipjobIsAnImport = False)
THEN
      BEGIN
            MoveGantryToShipEmpty := False;
            JobToBeDone := True;
            _MoveGantryToLandSide := False;
            TimeToMoveToShipFull := 60;
      END ELSE
IF (_MoveGantryToLandSide = True) AND (NextLandJobIsAnImport = False)
THEN
      BEGIN
            MoveGantryToLandEmpty := True;
            JobToBeDone := True;
            _MoveGantryToShipSide := False;
            TimeToMoveToLandEmpty := 40;
      END ELSE
```

```
IF (_MoveGantryToLandSide = True) AND (NextLandJobIsAnImport = True)
THEN
      BEGIN
            MoveGantryToLandEmpty := False;
            JobToBeDone := True;
            _MoveGantryToShipSide := False;
            TimeToMoveToLandFull := 60;
      END
      ELSE JobToBeDone := False;
IF (NumbStackRmgIdle > 0) AND (JobToBeDone = True) THEN
      STARTNEWJOB := True ELSE StartNewJob := False ;
END;


FUNCTION StartShoreSideWork : Boolean;
BEGIN
IF NextJobIsAnImport = True THEN
      BEGIN
            JobOutstanding := True;
            LoadImportFromImv := True;
            UnloadExportToImv := False;
            _Time := 60;
      END ELSE
      BEGIN
            JobOutstanding := True;
```

```
          UnloadExportToImv := True;
          _Time := 60;
          LoadImportFromImv := False;
     END;
IF (NumbRmgAtSSide > 0) AND (JobOutstanding = True) THEN
     STARTSHORESIDEWORK := True ELSE StartShoreSideWork := False;
END;


FUNCTION StartLandSideWork : Boolean;
BEGIN
IF NextJobIsAnImport = True THEN
     BEGIN
          JobToBeDone := True;
          LoadVehicleWithImport := True;
          UnloadExportFromVehicle := False;
          _Time := 60 ;
     END ELSE
     BEGIN
          JobToBeDone := True;
          UnloadExportFromVehicle := True;
          _Time := 60;
          LoadVehicleWithImport := False;
     END;
IF (NumbRmgAtLSide > 0) AND (JobToBeDone = True) THEN
     STARTLANDSIDEWORK := True ELSE StartLandSideWork := False ;
END;
```

```
FUNCTION StartMoveFromSSide : Boolean;
BEGIN
IF NextJobIsAnImport = True THEN
      BEGIN
            NextJob := True;
            MoveImportToStack := True;
            MoveEmptyToStack := False;
            _Time := 60;
      END ELSE
      BEGIN
            NextJob := True;
            MoveEmptyToStack := True;
            _Time := 40;
            MoveImportToStack := False;
      END;
IF (RmgToLeaveSSide > 0) AND (NextJob = True) THEN
      STARTMOVEFROMSSIDE := True ELSE StartMoveFromSSide := False ;
END;


FUNCTION StartMoveFromLSide : Boolean;
BEGIN
IF NextJobIsAnImport = True THEN
      BEGIN
            JobFound := True;
            MoveEmptyToStackFromGate := True;
            MoveExportToStack := False;
```

```
          _Time := 40;
      END ELSE
      BEGIN
          JobFound := True;
          MoveEmptyToStackFromGate := False;
          _Time := 60;
          MoveExportToStack := True;
      END;
IF (RmgToLeaveLSide > 0) AND (JobFound = True) THEN
      STARTMOVEFROMLSIDE := True ELSE StartMoveFromLSide := False ;
END;



FUNCTION LetTruckInPort : Boolean;
BEGIN
IF NumTruckOutside > 0 THEN TruckOutSide := True ELSE
      TruckOutside := False;
IF (ShipImportJobs > 0) OR (ShipExportJobs > 0) THEN
      shipJobsLeft := True ELSE ShipJobsLeft := False ;
IF shipJobsLeft = True THEN waitingForTruck := True ELSE
      WaitingForTruck := False ;
IF (truckOutside = True) AND (waitingForTruck = True) THEN
      LetTruckInPort := True ELSE LetTruckInPort := False ;
END;
```

```
FUNCTION MoveToBay : Boolean ;
BEGIN
IF NumbTruckToBay > 0 THEN
      BEGIN
            truckWaiting := True;
            Bay := WhichBay ;
      END ELSE
      BEGIN
            truckWaiting := False;
            bayFree := False;
            Bay := 0 ;
      END;
IF Bay > 0  THEN
      BEGIN
            bayFree := True;
            BayPos := Bay;
      END ELSE bayFree := False;
IF (truckWaiting = True) AND (bayFree = True) THEN
      BEGIN
            MOVETOBAY := True;
            _Time := 0;
      END ELSE MOVETOBAY := FALSE ;
END;
```

```
FUNCTION StartShipArrive : Boolean ;
BEGIN
 IF (NumberOfShipsAtSea > 0) AND (ShipArrivalDue = True) THEN
      BEGIN
            STARTSHIPARRIVE := True ;
            _Time := 1200 ;
      END ELSE STARTSHIPARRIVE := False;
END;



FUNCTION StartDockAtBerth : Boolean;
BEGIN
  IF (NumbShipsWaitingToBerth > 0) AND (NumbFreeBerths > 0) THEN
      BEGIN
            STARTDOCKATBERTH := True;
            _Time := 120;
      END ELSE STARTDOCKATBERTH := False;
END;


FUNCTION StartShipLeave: Boolean ;
BEGIN
    IF WorkAtBerthCompleted = True THEN
      BEGIN
            STARTSHIPLEAVE := True;
            _Time := 3600;
      END ELSE STARTSHIPLEAVE := False;
END;
```

```
FUNCTION SendEmptyImvToStore : Boolean;
BEGIN
      IF StoreNumWithExpContainer = 0 THEN noExportsInStore := True ELSE
            NoExportsInStore := False ;
      IF ExportsLeftForShip = 0 THEN shipFullOfExports := True ELSE
            ShipFullOfExports := False;
      IF (shipFullOfExports = True) OR (noExportsInStore = True) THEN
            _MakeImvIdle := True ELSE _MakeImvIdle := False ;
      IF (NumbEmptyImvsToStore > 0) AND (_MakeImvIdle = False) THEN
      BEGIN
            SENDEMPTYIMVTOSTORE := True;
            _Time := 30;
      END ELSE SENDEMPTYIMVTOSTORE := False;
      {SENDEMPTYIMVTOSTORE Must return _MakeImvIdle}
END;


FUNCTION SendFullImvToStore : Boolean ;
BEGIN
      IF NumbFullImvsToStore > 0 THEN
      BEGIN
            SENDFULLIMVTOSTORE := True;
            _Time := 60;
      END ELSE SENDFULLIMVTOSTORE := False;
END;
```

```
FUNCTION SendEmptyImvToShip: Boolean;
BEGIN
      IF (NumbEmptyImvToShip > 0) THEN
      BEGIN
            SENDEMPTYIMVTOSHIP := True;
            _Time := 60;
      END ELSE SENDEMPTYIMVTOSHIP := False;


END;



FUNCTION SendFullImvToShip : Boolean ;
BEGIN
      IF (NumbFullImvToShip > 0) THEN
      BEGIN
            SENDFULLIMVTOSHIP := True;
            _Time :=120;
      END ELSE SENDFULLIMVTOSHIP := False;
END;
```

```
{CRANEJOBS must return _LoadImv, _UnLoadImv, _LoadShip, _UnloadShip,
FetchImvFromShipQ, FetchImvFromOtherShipQ, FetchImvFromIdleQ,
_FullImvToUnload, ReturnImvsToIdle, ChangeACraneToExports}

FUNCTION CraneJobs : Boolean;
BEGIN
    IF (ShipInBerthOne = True) OR (ShipInBerthTwo = True) THEN
        berthedShip := True
    ELSE BerthedShip := False ;
    IF (berthedShip = False) AND (NumEmptyImvAtShip > 0) THEN
    BEGIN
        imvsIdle := True;
        ReturnImvsToIdle := True;
    END ELSE imvsIdle := False;
    IF BerthedShip = True THEN
    BEGIN
        IF NumEmptyImvAtBerth > 0 THEN
        BEGIN
            emptyImvToLoad := True;
            FetchImvFromShipQ := True;
            FetchImvFromOtherShipQ := False;
            FetchImvFromidleQ := False;
            TimeToFetchImvFromShipQ := 40;
        END ELSE
        IF NumEmptyImvAtOtherBerth > 0 THEN
        BEGIN
            emptyImvToLoad := True;
            FetchImvFromOtherShipQ := True;
            FetchImvFromShipQ := False;
            FetchImvFromIdleQ := False;
            TimToGetImvFromOtherShipQ := 40;
        END ELSE
        IF NumIdleImvs > 0 THEN
        BEGIN
            emptyImvToLoad := True;
            FetchImvFromIdleQ := True;
            FetchImvFromShipQ := False;
            FetchImvFromOtherShipQ := False;
            TimeToFetchImvFromIdleQ := 40;
        END ELSE emptyImvToLoad := False;
END;
IF BerthedShip = True THEN
        IF NumFullImvAtBerth > 0 THEN
        BEGIN
            _FullImvToUnLoad := True;
            TimeToUnloadFullImv := 40;
        END ELSE
            _FullimvToUnload := False;
IF BerthedShip AND (CraneOnImports = True)
        AND (CraneLoaded = True) AND (emptyImvToLoad = True)
        THEN _LoadImv := True
 ELSE _LoadImv := False ;
IF BerthedShip AND (CraneOnImports = False)
        AND (CraneLoaded = False) AND (_FullImvToUnLoad = True)
```

```
      THEN _UnLoadImv := True
ELSE _UnLoadImv := False ;
IF BerthedShip AND (CraneOnImports = False) AND
      (CraneLoaded = True) THEN
BEGIN
      _LoadShip := True;
      TimeToLoadShip := 40;
END ELSE _LoadShip := False;
IF (CurrentShipBerth = 1) AND
      ((NumImportsRemaining < (NumTotalImportJobs / 2)) AND
      (NumCranesOnImports = 2)) OR
      ((NumImportsRemaining = 0) AND
      (NumCranesOnImports = 1)) THEN
BEGIN
      changeACraneToExports := True;
      CraneOnImports := False;
END;
IF (CurrentShipBerth = 2) AND
      ((NumImportsRemaining< (2*NumTotalImportJobs/3)) AND
      (NumImportsRemaining> (NumTotalImportJobs/3)) AND
      (NumCranesOnImports=3)) OR
      ((NumImportsRemaining< (NumTotalImportJobs/3)) AND
      (NumImportsRemaining > 0) AND
      (NumCranesOnImports=2)) OR
      ((NumImportsRemaining = 0) AND
      (NumCranesOnImports = 1)) THEN
BEGIN
```

```
          changeACraneToExports := True;
          CraneOnImports := False;
     END ELSE
          changeACraneToExports := False;
     IF BerthedShip AND (ChangeACraneToExports = False) AND
          (CraneOnImports = True) AND
          (CraneLoaded = False) THEN
     BEGIN
          _UnLoadShip := True;
          TimeToUnloadShip := 40;
     END ELSE _UnloadShip := False;
     IF ((CraneOperational = False) AND (imvsIdle = True)) OR
               (CraneOperational = False) THEN CRANEJOBS := False;
     IF ((_LoadImv = True) OR (_UnLoadImv = True) OR
          (_LoadShip = True) OR (_UnLoadShip = True)) AND
          (NumShipCranesIdle > 0) THEN CRANEJOBS := True ELSE
          CraneJobs := False ;
END;
END.
```

APPENDIX E

CONTENTS - APPENDIX E

APPENDIX E

DESIGNER - An interactive approach to man/machine interface development.

E.1 INTRODUCTION

Designer was created as a tool for use with the ESSIM simulation language and expert system, allowing programmers to add a graphic/windowing interface for the presentation of output. The concept behind Designer was not just to provide a library of pre-written routines, but to let the user create an interface interactively. Designer is a form of 4GL in which 'interactive programming' is used to generate PASCAL program code. Once created, the interface layout can be 'edited' and new code produced. Simulation models which initially provide a crude form of input/output and screen design can be transformed by replacing 'read' and 'write' commands. Furthermore, the eventual user of the model can be directly involved with the setting out of the interface and the presentation of the output.

E.2 USING DESIGNER

The standard Designer interface is based on the use of high-resolution inverse-video graphics (black characters on a white background). Characters shapes are user defined and options are selected using a mouse. All input and output, whether in graphic or character format, is displayed in 'pull-down' or 'pop-up' windows. The top two lines of the screen are reserved for default menu options. The bottom line is used for the display of instructions.

The default menu options are specified by simply typing the appropriate text. Two or more spaces indicates the start of a new option. The position of the menu options is automatically adjusted such that an even layout is always obtained. Pointing the mouse icon at an options results in its display characteristics being reversed.

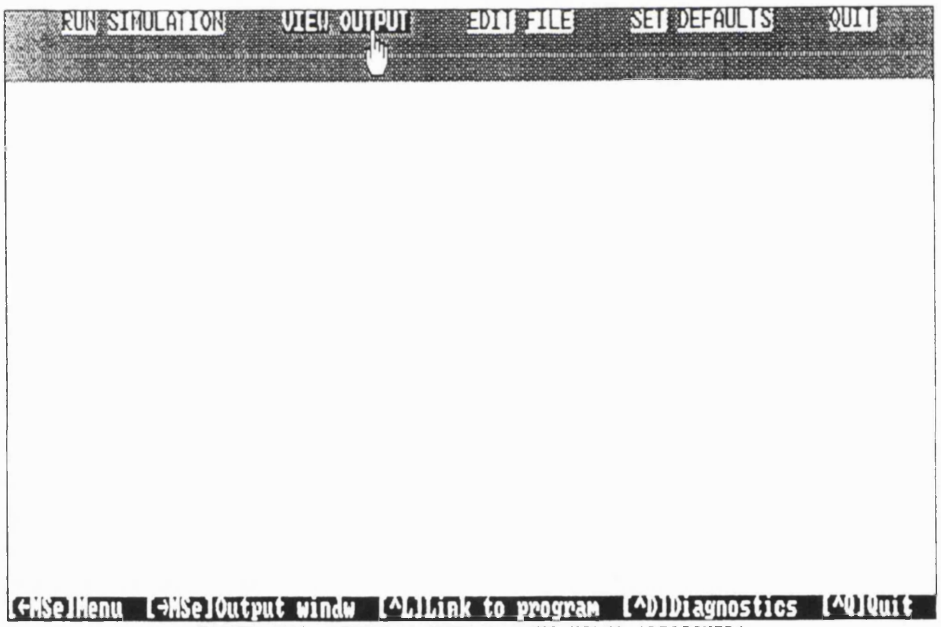


FIGURE E76 DEFINING MENU OPTIONS USING 'DESIGNER'

E.2.1 CREATING PULL-DOWN WINDOWS - Pointing the mouse icon at a default option and clicking the left mouse button results in the creation of a pull-down window. The width of the window is set to be the same as that of the displayed option. The height of the default window is sufficient for a single window option. Window options are entered by simply typing the appropriate text. If the width of the text exceeds the width of the window, the window size is adjusted. Pressing the enter key expands the window downwards and positions the cursor on the following line. Text input to the window is terminated by pressing the ESC key. Pointing the mouse icon at a window option reverses the display characteristics of the text. The window is removed from the screen by moving the mouse icon to a point outside the window area.

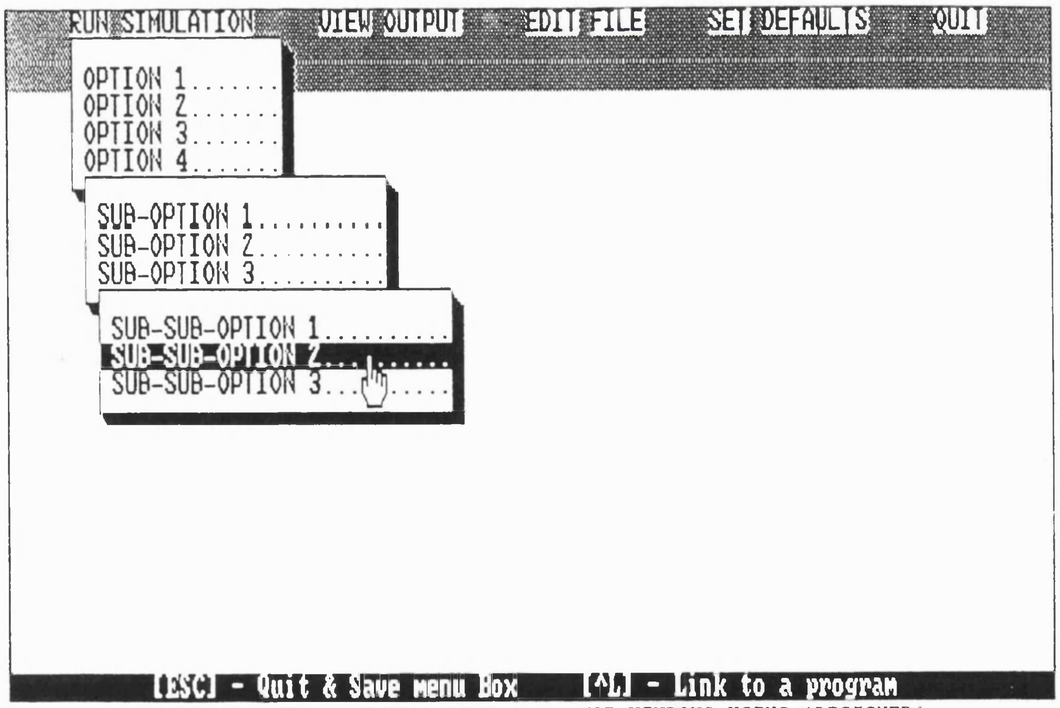
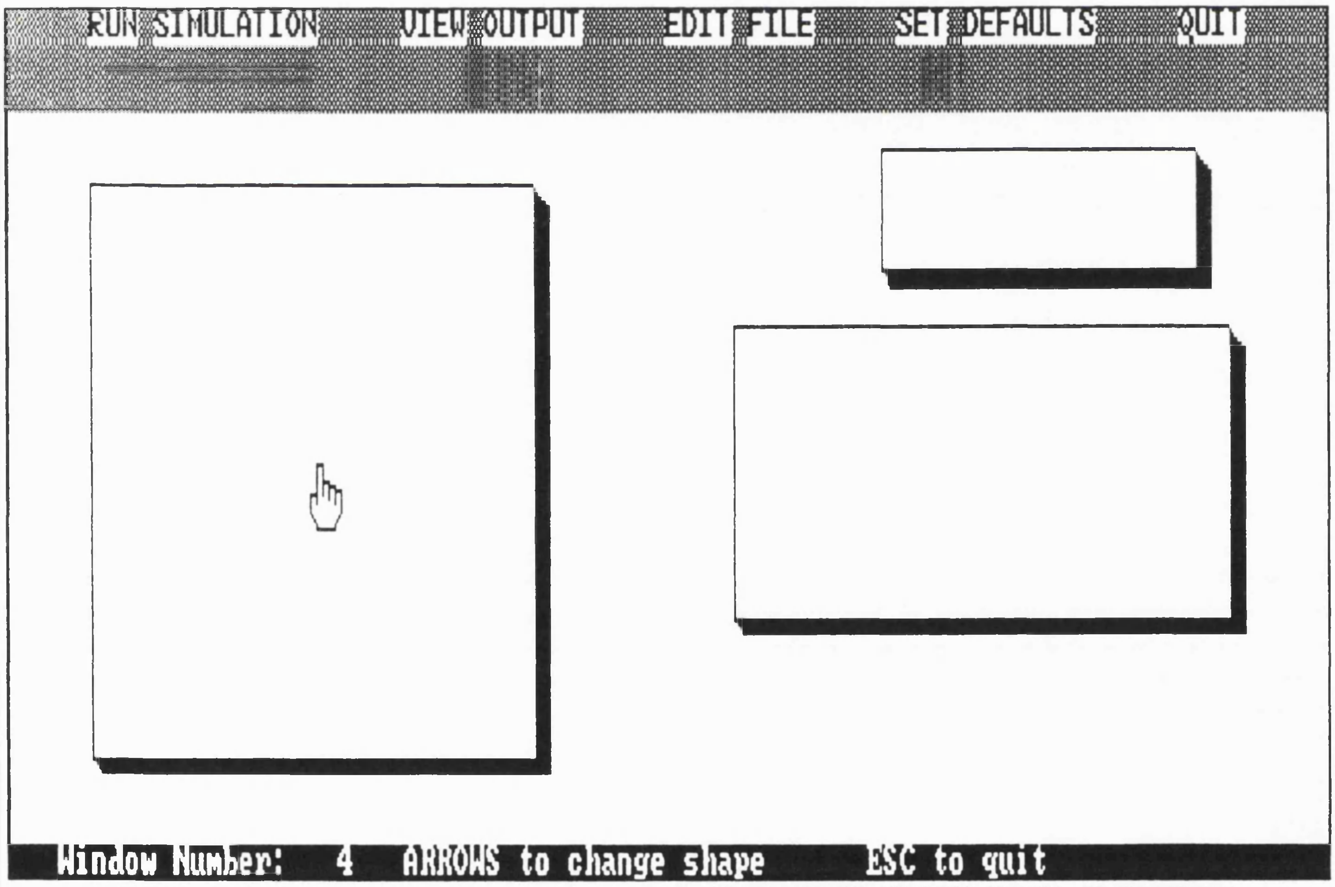


FIGURE E77 CREATING PULL-DOWN MENU TYPE WINDOWS USING 'DESIGNER'

Pull-down windows can be stacked. Pointing at a new window option and clicking the left mouse button results in the creation of a default window. Window options and further sub-windows can be specified as before.

E.2.2 POP-UP WINDOWS - Pop-up windows differ from pull-down windows in that they are independent from any specific menu option. Furthermore, pop-up windows are not used for the display of menus but rather for the display of free-form text, requests for user input, and the creation of graphical forms.



Window Number: 4 ARROWS to change shape ESC to quit

FIGURE E78 CREATING POP-UP WINDOWS FOR SIMULATION OUTPUT USING 'DESIGNER'

Pop-up windows are created by clicking the right mouse button. The window can be positioned by moving the mouse, and the size of the window altered using the arrow keys. A code unique to the new window is displayed at the bottom of the screen. The window 'editing' session is terminated by pressing the ESC key. Several pop-up windows can be displayed simultaneously to make it simpler to sort out relative positioning problems.

E.2.3 EXTERNAL PROGRAM CALLS - External programs can be activated by associating a file name with a menu option. Designer remains memory resident until termination of the sub-process. To specify a file name, the mouse icon has to be positioned over a given window option. Typing CTRL-L leads to the user being guided through a series of questions. Several types of program calls are possible. The user can 'chain' or 'spawn' a program, with, or without the use of parameters. Alternatively DOS commands can be activated, again with the possibility of parameters passing. Output generated by external programs can either be displayed on a clear screen(in text-mode) or re-directed to a specified pop-up window. What is particularly unusual with Designer is that the program calls can be tested straight away without having to generate and compile the code. Designer's ability at executing DOS commands is particularly useful in the context of providing the eventual user, facilities such as directory listings (possibly to a window), file copying/backup, changing default directories, and so on.

E.2.4 CODE GENERATOR - Having designed the interface, the corresponding program can be produced by typing CTRL-Q . The user is prompted for a file name. Designer then generates the PASCAL code and compiles it to EXE format. A 'screen design' file is also generated which can be used to re-load a previously designed interface.

E.3 MODIFYING DESIGNER FILES

There are two ways of modifying Designer files. Re-loading the screen design file or altering the PASCAL code. The PASCAL code is split into modules. Only one of these modules is accessible by the user and consists of a single procedure. The procedure consists of a CASE statement relating to window and menu options. By inserting library commands and/or procedure and function calls, particular menu options can be made to activate given tasks. The file must then be re-compiled using the RERUN batch file to analyse the effect of the changes.

E.3.1 USING THE GENERATED FILES - Having designed an interface, the user's program has to be modified. The first step is to alter the heading of the program code from PROGRAM FileName to MODULE FileName. The second step consists in enclosing the program body within a procedure. This procedure must then be called by one of the menu options. This can be done by modifying the CASE statement as described above. The procedure must also be declared as being EXTERNAL before LINKING the modules. The read and write statements in the user's program will also need changing. These can be replaced by READWIN and WRITEWIN library functions that re-direct I/O commands to specified windows. Graphs can also be produced within windows using the appropriate library functions. All the ESSIM port model interfaces were created using Designer.

APPENDIX F

CONTENTS - APPENDIX F

Page

APPENDIX F

OBJECT-ORIENTED SIMULATION

F.1 INTRODUCTION

Knowledge Base Systems (KBS) and Expert Systems (ES) have been receiving increasing attention as potential components of a new generation of simulation models. The interest in amalgamating the two techniques has resulted predominantly out of the difficulties in representing human expertise using present simulation modelling methods. Furthermore, ESs have increasingly been applied to problems of manufacturing control in real-time which has resulted in the need to represent the same ES components in a simulation model of the real-world system.

A number of approaches have been used in combining expert system and simulation methodologies in creating an environment for decision support. The method that will be discussed is that of Object Oriented Programming (OOP) in which human expertise in the form of rules depicting decision making is represented as an integral part of the simulation model.

The next section will consider the general features of Object Oriented languages. In section F.3, the application of the object oriented approach will be considered with respect to a number of developments which have use the OOP concept in developing simulation models (other than F.3.1 which discusses the Smalltalk language). In section F.4, the advantages and disadvantages of the OOP approach are considered and compared to alternative methodologies.

Some brief concluding remarks are given in section F.5.

F.2 WHAT IS OBJECT ORIENTED PROGRAMMING

Object-Oriented programming is a fairly loose term to describe a method of knowledge representation based on the description of objects and their interrelationships. The technique originates from the AI field where it is used in developing Expert Systems most often using LISP as a basis for the language construct. It was not long before it was noticed that the Object-Oriented approach would be suitable for the development of simulation models. Being based on the Expert System(ES) paradigm, object-oriented simulation provides an effective environment for the specification of domain knowledge.

In constructing an object-oriented simulation, the user first creates a set of objects that broadly correspond to real-world objects. The characteristics of these objects are then defined; the inputs they respond to, and the actions they carry out in response. The interplay between objects is represented by the passing of messages. In other words, the action carried out by one object may lead to a message being transmitted to another object specifying that an action should be carried out.

Another important aspect of object-oriented simulation is the concept of 'inheritance' which is derived from the semantic networks knowledge representation scheme used in many expert systems. Inheritance is useful in creating hierarchies of objects, each of which can inherit characteristics from a higher ranking set.

F.3 EXAMPLES OF LANGUAGES BASED ON THE OBJECT ORIENTED APPROACH

F.3.1 SMALLTALK-80 (Reference Ulgen and Thomasma[1986])

The Smalltalk-80 language was developed about 15 years ago to provide an alternative to the procedural programming techniques. Smalltalk-80 replaces the procedural concept of operators and operands by that of messages and objects. A good example described by Ulgen and Thomasma[1986], is a comparison of a mathematical operation using a procedural and object-oriented language. In the case of a procedural language, an operation such as SIN(X) would be carried out by applying the SIN operator on the operand. The operand remains passive whereas the operator is active and carries out a calculation based on the value of the operator. The integrity of the operation is maintained by insuring that the operand is of the correct data-type.

In contrast, the object-oriented approach defines the operator (X) as being an object. The object can then perform the operation (SIN) on itself. Consequently, a more logical way of writing the command would be X SIN in which SIN is a message sent by the object (X) asking for the operation to be performed. In Smalltalk-80, messages that do not have arguments are referred to as unary messages. Conversely, messages that have one or more arguments are known as keyword messages. The syntax of a typical message that has two arguments would be as follows:

Machine acquire: 1 Resource: 'replacement grinder'

Machine is the object that receives the message. The two arguments are 1 and 'replacement grinder'.

Inheritance also plays a role in such a simple operation. Smalltalk-80 supports a tree like structure for the classification of objects and operations. For example, X may belong to a predefined 'Class' of objects known as 'Number'. 'Number' is in turn known to consist of three subclasses; Float, Fraction and Integer. The class 'Integer' in turn has three subclasses; SmallInteger, LargePositiveInteger, and LargeNegativeInteger. These classes are arranged in a hierarchical order in which a subclass is contained entirely within its superclass. Consequently, an instance of a specific class must also be an instance in the corresponding superclass. Operations are also defined as part of classes.

The same class systems applies to operations. For example, Factorial belongs to the class Integer. If the message Factorial is sent to an instance of the class SmallInteger (say X), then a search for the Factorial operation will first be made in the SmallInteger class and if it is not found, then in its superclass, Integer. If this were also to be unsuccessful the backward search would continue until no more superclasses were left (The top class is 'Object' and is the only class not to have a superclass).

Smalltalk-80 is a general purpose object-oriented language and is therefore not specifically designed for simulation. As with any high level procedural language such as Pascal or C, the simulation control framework has to be defined. Indeed, Smalltalk-80 can be used to create models that use any of the well known methodologies such as activity scanning, process interaction or event scheduling. Furthermore, the language can be used to model time using the methods of discrete changes, continuous changes or combined discrete/continuous changes.

ADVANTAGES	DISADVANTAGES
Compact code.	Code difficult to understand.
Supports inheritance.	Not designed for simulation.
graphical output possible.	Difficult to debug.
	Data structures limited.
	Inflexible (similar to 4GL)

F.3.2 ROSS (Klahr[1984], Klahr[1985], Klahr & Faught[1980], McArthur[1986], Adelsberger [1986])

ROSS (Rand Object-oriented Simulation System) is an object-oriented simulation language that uses an English like syntax which is supposed to make the code easier to read and intelligible to non-programmers. ROSS is implemented in LISP (MacLisp, Interlisp-20, Vax-Interlisp, Interlisp-D, Franzlisp and Zetalisp) and is therefore interpreted. This, RAND claim, is an advantage since it permits interruption of the simulation run (for queries & code modification), removes the need for compilation (hence easier & quicker modification/experimentation cycle), and simplifies the debugging process. ROSS also provides textual simulation output designed for tracing purposes and graphics facilities for animated presentations.

ROSS differs from Smalltalk-80 in that it is specifically designed for simulation, but the general concept of programming using objects and message passing is retained. The syntax for a message being passed from one object to another is as follows:

```
(ASK <object> <message>)
```

for example:

```
(ASK operator1 send repair_team1 to machine3)
```

<message> is a sequence of LISP ATOMS. In the example, 'operator1' is sent the message 'send repair_team1 to machine3'.

Having sent a message to an object, the object has to have a way of responding. In ROSS, the response to the message will also take the form of messages and has the following syntax:

```
(ASK <object> WHEN RECEIVING <message-template> <body>
```

<message-template> is a sequence of Lisp atoms uses in detecting the message being received. <Body> is a list of ROSS commands or LISP code. An example object response would be:

```
ASK operator WHEN RECEIVING (send >repair_team to >machine)
        (ASK !myself add !repair_team to list of busy_teams)
        (ASK !myself remove !repair_team from list of
        repair_teams_available)
```

In ROSS, the concept of global functions is not supported and so each object in the system must have a behaviour defined (it is assumed that all objects in the system are unique). In the above example, when operator1 receives a message, it gets compared to the message template. The symbol > which is used as a prefix in the message template indicates the use of a variable. For example, >repair_team is a variable and will be bound to its value 'repair_team1'. Variables in <body> are prefixed by the character ! which indicates that the value of the variable should be used rather than its name. Also notice the use of the variable !myself. This indicates that a message should be passed to the currently active object (a form of looping with a rather strange syntax). Hence, a typical list of actions carried out by an object consist of a complex chain of subsequent message transmissions (which include messages passed from an object to itself).

As with Smalltalk, ROSS supports the concepts of object hierarchies and inheritance. To syntax for creating a new object is as follows:

```
        (ASK <object1> CREATE GENERIC <object2>)
```

for example:

```
        (ASK operator CREATE GENERIC operator1)
```

'Operator1' is created as being an instance of 'operator'. In this case, 'operator' can be interpreted as a class of object while 'operator1' represents a member of that class. 'operator1' inherits the attributes of its parent which is why in the first example a message was sent to the object 'operator1' and yet in the subsequent example, the behaviour of the object was attributed to the object 'operator'. However, the fact that 'operator' is a subclass of 'operator1' does not mean that their behaviour must be identical. During code execution, ROSS first checks whether 'operator1' has any behavioural responses attached to it. If none can be found, the parent class of which the object is an instance is searched.

A simulation model known as SWIRL, developed using the ROSS language (McArthur[1986]) is a good example of the use of inheritance structures. When an instance of an object of the hierarchy receives a message, a search is made through the hierarchy to find an object that has a behaviour attributed to it that matches the message received. So for example, if the object 'fighter-base1' receives a message, a successive search is done through the defined object behaviours for 'fighter-base1', 'fighter-base', 'fixed-object' and finally 'simulator' (top class in the hierarchy). The objects or classes of object can also have values associated with them. For example, in creating 'fighter-base' the following CREATE command could be used:

```
(ASK fixed-object CREATE GENERIC fighter-base WITH
            Position              (0 0)
            Status                active
            Fighters-avail        nil
            Scramble-delay        10
            Alert-duration        1800
```

some of the values such as 'Alert-duration' are taken as being defaults for all instances of 'fighter-base'. On the other hand attributes such as 'Fighter-avail' are given a NIL value meaning that each individual instance of 'Fighter-base' may have its own value for the specified attribute. The values of attributes, as with variables in general, can be manipulated as follows using messages:

```
(ASK Fighter-basel RECALL YOUR alert-duration)

(ASK Fighter-basel SET YOUR status TO destroy)

(ASK Fighter-basel INCREMENT YOUR alert-duration BY 100)
```

In the second and third example, if 'status' or 'alert-duration' do not exist for the specified object, then the attribute is automatically created. In the third example, if 'alert-duration' does not exist its value is inherited from the 'fighter-base' class, the attribute created under 'fighter-base1' and the value 100 added to its present value (1800).

ROSS provides commands that are specific to simulation. Time handling is of course one of the most critical aspects of simulation and is handled by commands of the following form:

```
(ASK operator3 PLAN AFTER 20 SECONDS
      TELL Machine2 to terminate)
```

the 'PLAN AFTER' command ensures that the following message is only sent after the specified delay in simulated time has elapsed. To advance clock time, ROSS provides the following command:

```
(ASK nclock TICK)
```

'nclock' has an attribute 'ticksize' which determines the clock time increment. Advancing the clock by a specified number of time units does not mean than an event that should have occurred before the new clock time does not get executed. The events that have been 'missed' are carried out until the new clock time has been reached. This simply permits interruption of the simulation at set time intervals for analysis of results.

ADVANTAGES AND DISADVANTAGES OF ROSS

ADVANTAGES:

1. English like syntax increases the readability of the code.
2. Some simulation problems can naturally be expressed in terms of objects and messages. In particular human communication, which explains why object-oriented programming originates from Artificial Intelligence research.
3. High level code which simplifies the programming task.
4. Good for rapid prototyping because of the interpreted natures of the code.
5. ROSS uses just a few commands which simplifies the programming process (ASK & TELL are the most common instructions). In this respect the language resembles LISP, in which ROSS is itself written.
6. 'Inheritance' can simplify the programming task if objects in the system have similar characteristics.

DISADVANTAGES:

1. The user has very little control over variable structures and data types.
2. ROSS assumes that the messages that are going to be sent are simple consisting of just a few variable values. In most cases, real-world messages are far more complex than can be modelled.
3. 'Inheritance' though useful in many cases, can make tracing execution of the model difficult. Complications are for example, inevitable when variables can assume values without there being any direct instruction from the programmer.
4. Some ROSS commands are unnecessary, messy and confusing. For example, the need for an object to sent itself messages in modifying variable values.
5. The interpretive nature of the code combined with the pattern matching characteristics of the message passing system, mean that large sections of simulation code are very slow to execute, though admittedly, quicker to modify.
6. ROSS has limited applicability. There would be little point in using ROSS in simulating problems which do not have the characteristics of inheritance.
7. Some simulation processes can be identified with objects and messages. This is not however always the case and it can become somewhat confusing to call some hypothetical process an 'object'. Furthermore, a message is often not what one can visualise an object as sending, particularly when the 'object' is not really an object in the first place!

8. A point mentioned by McArthur[1986] is that ROSS cannot deal effectively with 'non-intentional' events. The example given is of a plane entering the range of a radar. The plane is hardly going to announce its position to the radar by sending it a message! The plane entering the radar range is a side effect of it flying its course and can therefore be categorises as a non-intentional side effect.

F.3.3 KBS / SIMULATION CRAFT (Reddy Et. Al.[1986], Baskaran Et. Al.[1986])

KBS (recently renamed as SIMULATION CRAFT) was developed at Carnegie-Mellon University and resembles ROSS in design. KBS is written in a Schema Representation Language(SRL) which is itself implemented in Franz Lisp. Objects in a KBS model are represented by 'Schemata' which are made up of 'Slots'. The slots contain data corresponding to physical limitations, event behaviour etc.. An example schema for a distribution centre would be as follows:

```
{{distribution-centre:
        CAPACITY:
        INVENTORY:
        SHIPMENT-TRANSIT-TIME:
        RECEIVE-ORDER-EVENT:"receive-order-rule"
              Range:(TYPE instance event-rule)
        RECEIVE-SHIPMENT-EVENT:"Receive-shipment-rule"
              Range:(TYPE instance event-rule)
        ADMINISTRATOR
              Range:(TYPE instance administrator)
        ORDER-TRANSIT-TIME:
              Default:0
        BACKORDERS:
        TOTAL-ORDERS:
        INVENTORY-COST}}
```

The entries in capital letters represent the slots whereas the entries, 'Range' and 'Default' are known as Facets. The range facet defines the type of value that may fill the slot. The default facet is used in providing a default value if a specified slot is empty.

The schemata that describe the model can be interrelated to form a network by using slot values as links. Furthermore, slots may inherit values from slots in other schemata.

```
{{event-24
        INSTANCE:event-notice
        FOCUS:D1
                comment:Focus of event,the entity
        EVENT:"receive-order-event"
                comment:event-slot in event-schema
        TIME:"21 April 1985 11:00:00"
                comment:time of execution
        PRE-ACTION:nil
                comment:Action to be taken before event execution
        POST-ACTION:nil
                comment:Action to be taken after event execution
        EVENT-PARAMETER:order10
                comment:Event parameters
        RUN-EVENT:run-event
                comment:method to execute event
}}
```

The above example is of a schema known as an event-notice. The event to take place at the specified time is the arrival of an order focused around the distribution centre D1. The "receive-order-event" entry is a cross reference to a slot in the 'focus' schema. The content of the slot may be a reference to a LISP function.

KBS uses the discrete event approach to simulation. Event behaviour can be expressed in the form of rules to be executed when the event occurs. The following example rule links with the distribution centre schema.

```
{{receive-order-rule
      INSTANCE:event-rule
      IF:(something-in-inventory)
      THEN:(schedule-transport)(deduct from inventory)
}}
```

There are other facilities provided by KBS which are meant to simplify the model building, experimentation and analysis stages. The user may build a rule base which can then be used in automatically selecting new experiments at the end of a simulation run. Furthermore, a facility exists for detecting causal relationships and defining these as part of the domain rule base.

Simulation Craft is a successor to KBS and is implemented using a knowledge engineering tool for the development of expert systems known as Knowledge craft. The underlying code in the system is written in Common Lisp. Although the kernel of simulation Craft is almost identical to that of KBS, Simulation Craft differs from KBS in a number of respects.

A Model Building Expert System can be used in assisting the user to create a graphical representation of the problem. The expert system also checks for model completeness and identifies inconsistencies. A Model Execution Expert is intended to help in deciding on start and end times for the simulation run as well as identifying the number of runs needed and the alternatives that should be evaluated. Finally, a Model Analysis Expert helps in evaluating the results of a simulation run.

Simulation Craft provides a further facility for automating the simulation life cycle. The Dynamic Planner is a rule based expert system that can be used in identifying a possible system configuration that could potentially be used to attain a desired goal. The expert system rule base consists of knowledge on cause and effect relations and on desirable system configurations.

ADVANTAGES & DISADVANTAGES OF KBS / SIMULATION CRAFT.

Because of the similarities between KBS and ROSS, the two environment have many of the same limitations. Where KBS differs is in the degree of background support provided to the user in developing and analysing the simulation. This additional support is based on the use of expert system knowledge-bases to analyse the available information. In this respect, the advice or decisions that are made automatically by KBS can only be as good as the coded knowledge. Whereas good advice given to the inexperienced simulation model user is unquestionably useful, one must also consider that bad advice is confusing and can be worse than providing no assistance at all. In the case of KBS/simulation craft, the problem is aggravated because of the degree to which the processes are automated.

F.3.4 BLOBS (REFERENCE: Middleton & Zanconato[1985])

As in the case of ROSS, BLOBS (BLackboard OBjectS) is an object-oriented language for simulation which was specifically developed for military aircraft control applications. BLOBS was developed using the POPLOG environment which is based on the AI language POP-11.

Middleton claims that it was originally intended to develop an environment based on an expert system that would obtain data from an existing radar simulator. The idea was abandoned in favour of an integrated approach because of the potential difficulties in sharing data from different sources and of modifying the behaviour of the simulation in response to a flight controllers decision represented by the expert system. The first system to be subsequently developed consisted of a blackboard system for modelling the aircraft controller and an object-oriented message passing system for representing the aircrafts and radars. Problems did however arise because the blackboard model held all data centrally with no possible restriction on access. There was therefore no way of ensuring consistency.

Objects in the model are described as sets of declarations and definitions each known as a BLOB. A BLOB may consist of local variable declarations, procedure definitions, behavioural responses (procedure-like) and an inheritance list.

Communication between BLOBS is possible in a number of ways: One BLOB may interrogate the public variables of another BLOB, a message may be sent to a BLOB, or demons may activate a behavioural response. Demons are attached to variables and may be activated by a change in the value of the variable. Demons can also monitor the creation and removal of BLOB instances.

The following is an example of the representation of an aircraft under BLOBS:

```
Dynamic blob aircraft;
    public vars position heading speed climb_rate;
    private vars target_heading target_height target_speed;
    on_message change_course
        with new_heading -> my target_heading do
            ;;; Initiate change in heading
    enddo;
    on_message climb
        with  new_height -> my target_height
              climb_rate -> my climb_rate do
        ;;;Initiate climb
    enddo;
    ;;;Other behaviours
endblob;
```

ADVANTAGES & DISADVANTAGES OF BLOBS

The author of the BLOBS system claims that the language has several advantages over the ROSS implementation: The distinction between local and global variables in defining objects, the use of demons in triggering behaviours when variables are updated, stricter type checking of messages and the possibility of deleting objects during the simulation run. On the other hand, the author also recognises certain pitfalls in the BLOBS language: A BLOB that is a recipient of a message requires prior knowledge of the identity of the sender. A BLOB that is created during the simulation run cannot have demons attached to it. And finally, the restrictions imposed by the type checking (and consequently the inability to apply arbitrary expressions), are said to cause problems with the generalisation of methods.

F.3.5 SLICE (Reference: Gosling & Okseniuk[1986])

SLICE (Simulation in LIsp of Continuous Events) is a LISP program code generator that is based on the object-oriented message passing paradigm. SLICE was originally developed for modelling air traffic control systems. SLICE represents the objects of the model as 'actors' and allows the user to define their behaviour in terms of continuous or discrete events. Information can be passed between actors in the form of messages and communication with a central database is also possible.

SLICE allows an expert system rule-base to interact with the model. This may be achieved by defining the expert system as an actor, or by representing it as a separate entity, invokable by the behaviour routines of actors.

SLICE can be used as a program generator using a 'script' file for the definition of the problem. Alternatively, SLICE functions can be embedded in existing LISP code.

Other SLICE properties are as follows:

- Data is local to an actor. Remote access is not possible other than by sending and receiving messages.
- A central database can be used by any of the processes, using messages to gain access. entry to specific areas of the database can be restricted.
- messages sent between actors contain certain specific information. A unique identification code, sender identification, recipient identification, start time, duration and content.
- Objects (and more specifically their defined behaviours) have different priority levels. In establishing the event queue, the system considers both the priority level and the scheduled time for activation. Messages are always taken as having the highest priority.
- Processes (known as T-processes) in SLICE contain data structures and function definitions. Furthermore, these processes may have many 'instances' of actors of the same type. These instances contain the same data and function definitions, but the value of the data items may differ. T-Processes are organised hierarchically and can inherit the characteristics of higher ranking processes (the structure is identical to that used in ROSS).

Each T-process consists of a stack (stores information on the execution state), a list of pointers linking variable names with their values, a reference to the message type expected, and a pointer to the t-process that is one level higher in the hierarchy of processes. The behaviour of a T-process is described by a set of IF-THEN type rules. The action part of the rules refer to functions that have the effect of advancing the simulation clock, sending messages etc..

The behaviour of actors may be inherited. Vehicles for example have some characteristics that apply to all vehicles, but also have features that are unique to a particular type of vehicle. In SLICE, T-processes are organised hierarchically with each sub-process being able to inherit the characteristics of its parent. This relieves the user from the burden of having to define general characteristics for every occurrence of a particular type of actor. Both data and rules can be inherited. The total data set for a specific T-process is made of the data specific to the T-process and the data values inherited from T-processes further up the hierarchy. However, when specific variables occur several times in different T-processes, the value corresponding to the lowest member of the hierarchy gets priority. In the case of rules, inheritance works slightly differently. Rules in different T-processes belonging to the same class of actor are combined. This leads to a risk of inconsistencies in the rules. However, rules may also be inconsistant for other reasons and so an algorithm is used in resolving the conflicts.

ADVANTAGES & DISADVANTAGES OF SLICE.

SLICE is different from the object-oriented languages previously discussed in that the systems is infact a LISP code generator. This, from the perspective of flexibility is an advantage. Another important benefit of SLICE is the importance attributed to modelling human behaviour in terms of IF-THEN type rules. The ability of accessing external rule-bases is also particularly attractive.

Facilities that are lacking include the ability to attach weights to different rules and the capability of seeking a goal in a rule-base.

F.3.6 SIMYON (References: Ruiz-Mier Et. Al.[1985,1987])

SYMION is a network simulation environment developed using the CAYENE language which is said to be based on a combination of object-oriented programming, logic programming and the discrete event approach to system modelling.

A SYMION model is created by unifying a number of CAYENE objects. These objects, are arranged hierarchically and can inherit the characteristics of objects in a parent class. SYMION also supports the use of DEMONS which are evaluated if an attempt is made at retrieving a specific value. For each defined object in the model, there exists a set of rules. When an object receives a message, a search is made of the rules to find a premise that matches the message type. The action part of the rule may then be carried out which typically involves sending messages to other objects. Consider the following message and rule as an example:

```
(send'CREATE'(Start))         Message

(Start) <== (Send_at >time_to_start MYSELF '(Next_arrival)) RULE
```

CREATE is the name of the object to which the message 'Start' is being sent. One of the rules which form part of the object definition has a premise that matches the message. CAYENE will now take the action part of the rule as a sub-goal. This results in a message 'Next_arrival' being sent to the object CREATE (represented by the MYSELF clause) at the time given by the constant 'time_to_start'. This new message also has a matching rule defined as part of the CREATE object:

```
(Next_arrival) <== (send_at TNOW >next_node
                          (newsym >transaction_name))
                   (send_at (+ TNOW >time_bet_creation)
                          MYSELF '(Next_arrival))
```

The action part of the rule now results in the creation of two further sub-goals which also take the form of messages. The use of >next_node in the first of two rules activates a demon requesting user input. A message is then sent to another object. The second rule is a next arrival scheduling mechanism which operates by scheduling the sending of a message by the object to itself (at time TNOW + time_bet_creation).

SYMION has a rule structure that is particularly attractive in the context of manufacturing problems. Consider the following object description:

```
(defob SCHEDULE
:properties:
(machines (M1 M2 M3..Mn))
:rules:
(Move ?Part ?Mach) <==
      (find ?Mach)
      (available ?Mach)
      (can_process ?Mach ?Part)
      (not(full(queue ?Mach)))
      (send_at TNOW ?Mach ?Part)
(Available ?Mach) <==
      (not(overloaded ?Mach))
      (not(down ?Mach))
(Down ?Mach) <==
      (maintenance ?Mach ?T1 ?T2)
      (lessp ?T1 TNOW ?T2)
(Down ?Mach) <==
      (needs_repair ?Mach)
```

The machines M1..Mn are objects that are defined as part of a class known as a SCHEDULE. The defined rules are used to check if a part (?PArt) should be moved to a particular machine (?Mach). The first rule represents the highest level of abstractions in the decision making process and can be translated as follows: Move part P to machine M if and only if M exists, M is available, M can process P, and if M's queue is not full. Each condition in the premise of the rule are treated as sub-goals and can be solved by other defined rules. For example, the availability of machine M is determined by checking if M is not currently overloaded and that it is not currently out of action. The search for solutions to sub-goals is not limited to the current object. For example, the rule for solving the 'Needs_repair' condition is defined as part of the object definitions:

```
(defob M1
        :properties:
        (a_Kind_of MACHINE)
        (queue Q1)
        (operator Operator1)
        (output_rate (/ ?Num_parts ?Time))
        (noise_level (sample_detector ?Det_1))
        :rules:
        (Needs_repair) <==
                (lessp >output_rate 13)
                (lessp 30 >noise_level)
```

The rule identifies machine M1 as needing repair if the output rate falls to below 13 and the noise level rises to above 13db.

ADVANTAGES & DISADVANTAGES OF SIMYON

SIMYON is a flexible language in terms of the use of rules. As Ruiz-Mier points out, changes to the logic of the simulation can be very simple to make. It would for example be very simple to include a rule that stipulates that an operator will stop working between certain times during the day:

```
(Busy Operator <== (lessp T1 TNOW T2)
```

Unfortunately, SIMYON does lack in power in processing defined rules. The inference strategy is purely backward chaining and as a consequence, the search for a solution will sometimes be very slow (when many sub-goals are identified). The provision of a forward chaining strategy would have been desirable. Other limitations exist but are less serious:

1. The syntax of the language is complex, in particular when defining mathematical statements.
2. Variable declarations are not required and so the risk of syntax errors in variable names are high.
3. Reading the code can be difficult as rules can be defined as part of different objects.

F.4 APPLYING THE OBJECT-ORIENTED APPROACH TO MODEL DEVELOPMENT

From the examples, one can see that the applicability of the object-oriented paradigm very much depends on the target problem for which a model is to be developed. The approach to modelling is dependent on the entities in the model having a sufficiently close relationship as to be able to establish a hierarchy in which inheritance of characteristics can play a part. The use of inheritance is a key factor in reducing the complexity of the model by limiting the duplication of facts and rules about objects. It is also desirable for the problem domain to be of a type that can be naturally broken down into constituent 'objects' or 'actors' and in which communication plays a significant role. These characteristics are not vital to the model development process, but simplify the overall task by allowing a more natural visualisation of the real world system, that the model is meant to represent. Such considerations are behind the suitability of the object-oriented approach to the simulation of tactical warfare problems to which the ROSS, BLOBS and SLICE languages specifically address themselves. In military applications, aircrafts, tanks etc. are effectively described using inheritance. In the case of aircrafts one could have a class inheritance hierarchy that would be as follows:

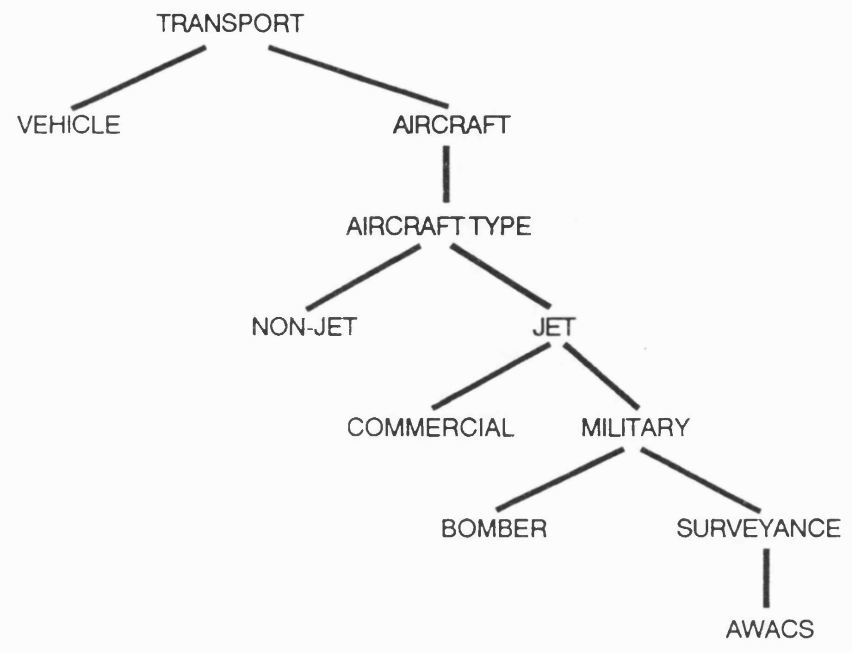


Figure F80 INHERITENCE IN OBJECT ORIENTED ENVIRONMENTS

Some of the characteristics of an AWACS plane are specific and cannot be found in any other existing military aircraft. These are therefore defined as part of the AWACS object description at the bottom of the hierarchy. However, some of the characteristics of an AWACS are more general and can therefore be inherited from classes of objects higher up in the hierarchy. Furthermore, an AWACS is used for surveyance operations and can therefore inherit characteristics from a separate hierarchy defining the capabilities of radar systems. Communication between object in a tactical warfare problem also plays an important role for which the object-oriented paradigm is well suited. An aircraft wishing to land at an airfield can for example be described as sending a message to the control tower requesting permission. The landing activity will then commence, conditional on the airstrip object being available.

Even in the case of applications that would seem suitable targets for an object oriented approach there can be problems such as those identified by McArthur[1986]. Of particular concern is the dependence on message passing for communication and activation of events. Consider the example of two enemy aircraft which are about to go into battle. For one aircraft to recognise and attack the other aircraft, messages need to be transmitted between the two which obviously contradicts the real-world rules of engagement.

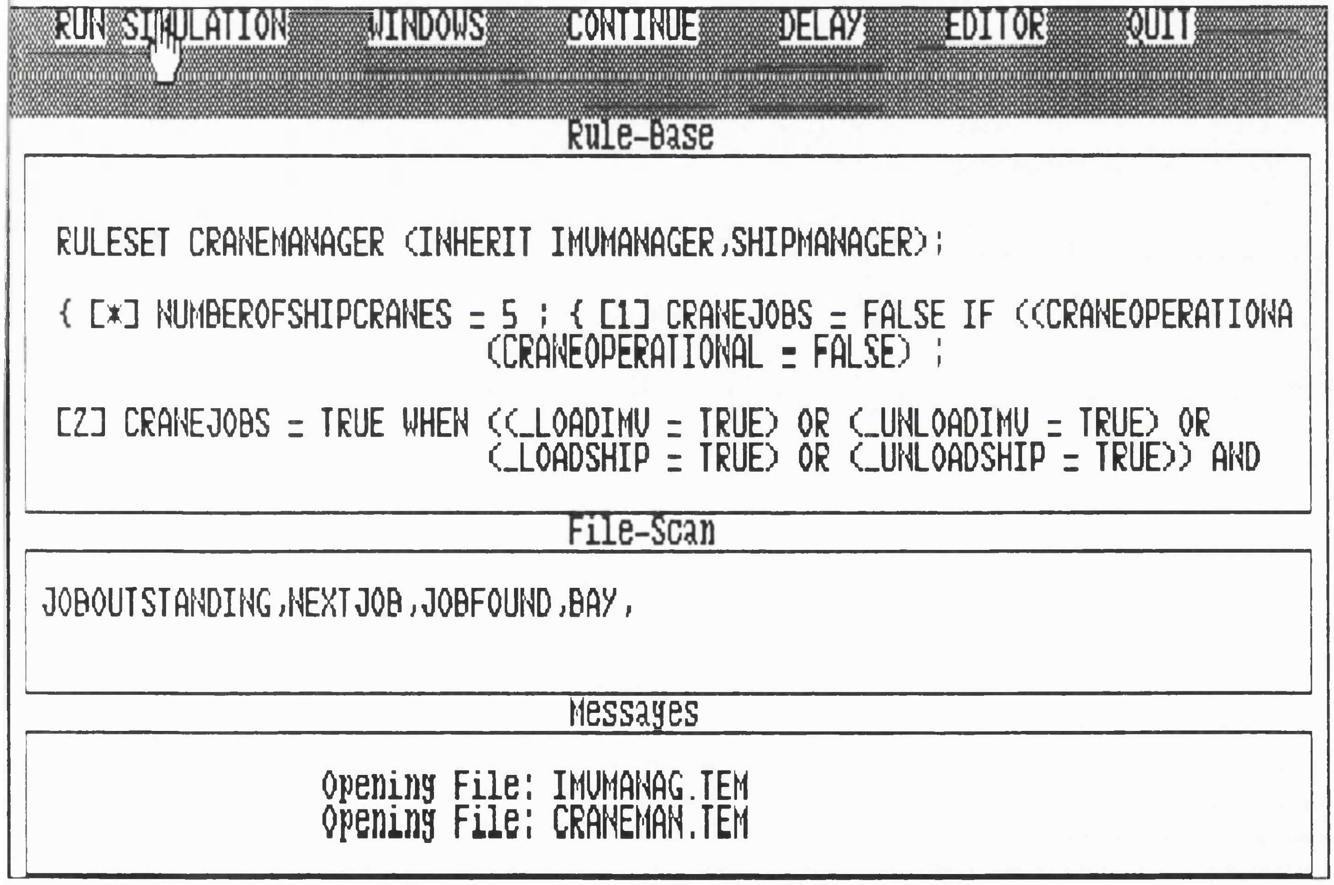
As with expert systems, models developed using the OOP approach are based on a relatively unstructured search algorithm. Furthermore, the desire to allow the user to develop the model incrementally by defining the characteristics of objects as and when they are identified also leads to problems in maintaining a structure. Consequent difficulties also arise because of the problem of ensuring that the defined model is complete and is not ambiguous or inconsistent. Lack of a formal structure also tends to mean that execution is slow for large models which is a problem aggravated by the interpretive nature of the Lisp environment which tends to be the language used in developing and implementing object-oriented models. The loss in speed is however offset by the advantages of being able to test the effect of changes in the code without having to compile and being able to trace and debug the model interactively.

The use of inheritance can be advantageous in terms of code size by reducing the repetition of characteristics of objects. However, problems can arise when values are inherited unexpectedly. Hence, the characteristics of each member of the object hierarchy has to be carefully defined with particular attention to the possible values that may be inherited from parent classes. Similar care is needed in OOP languages in which rules can be inherited. A set of rules may be spread across a number of object classes making it difficult to trace potential actions and increasing the risk of rules being mistakenly inherited in satisfying a goal. Such problems aggravate the difficulties in specifying the characteristics of the components of the model and particularly in cases where the concept of objects and messages do not seem to be a natural structure for the formalisation process.

APPENDIX G

APPENDIX G

ESSIM OUTPUT DISPLAYS FOR THE PORT MODEL

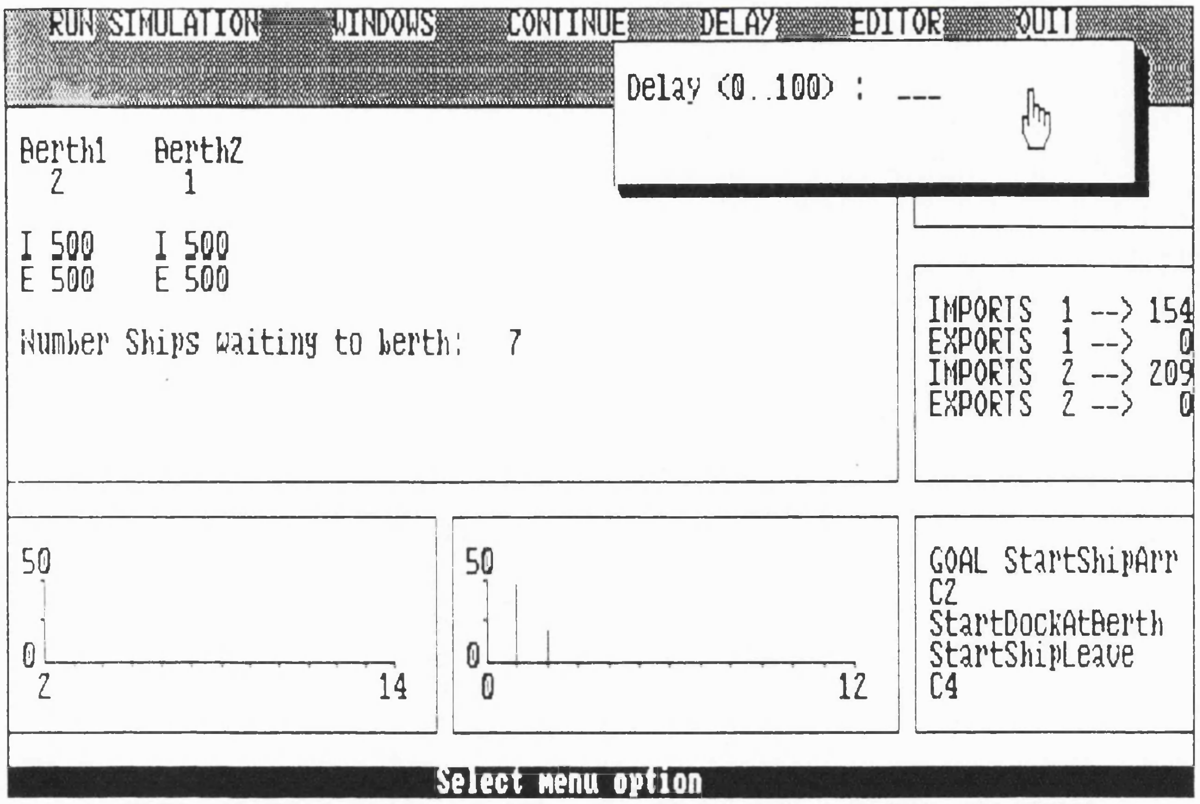


```
RULESET CRANEMANAGER (INHERIT IMVMANAGER,SHIPMANAGER);

{ [*] NUMBEROFSHIPCRANES = 5 ; { [1] CRANEJOBS = FALSE IF ((CRANEOPERATIONA
                                 (CRANEOPERATIONAL = FALSE) ;

[2] CRANEJOBS = TRUE WHEN ((_LOADIMV = TRUE) OR (_UNLOADIMV = TRUE) OR
                           (_LOADSHIP = TRUE) OR (_UNLOADSHIP = TRUE)) AND
```

```
JOBOUTSTANDING,NEXTJOB,JOBFOUND,BAY,
```

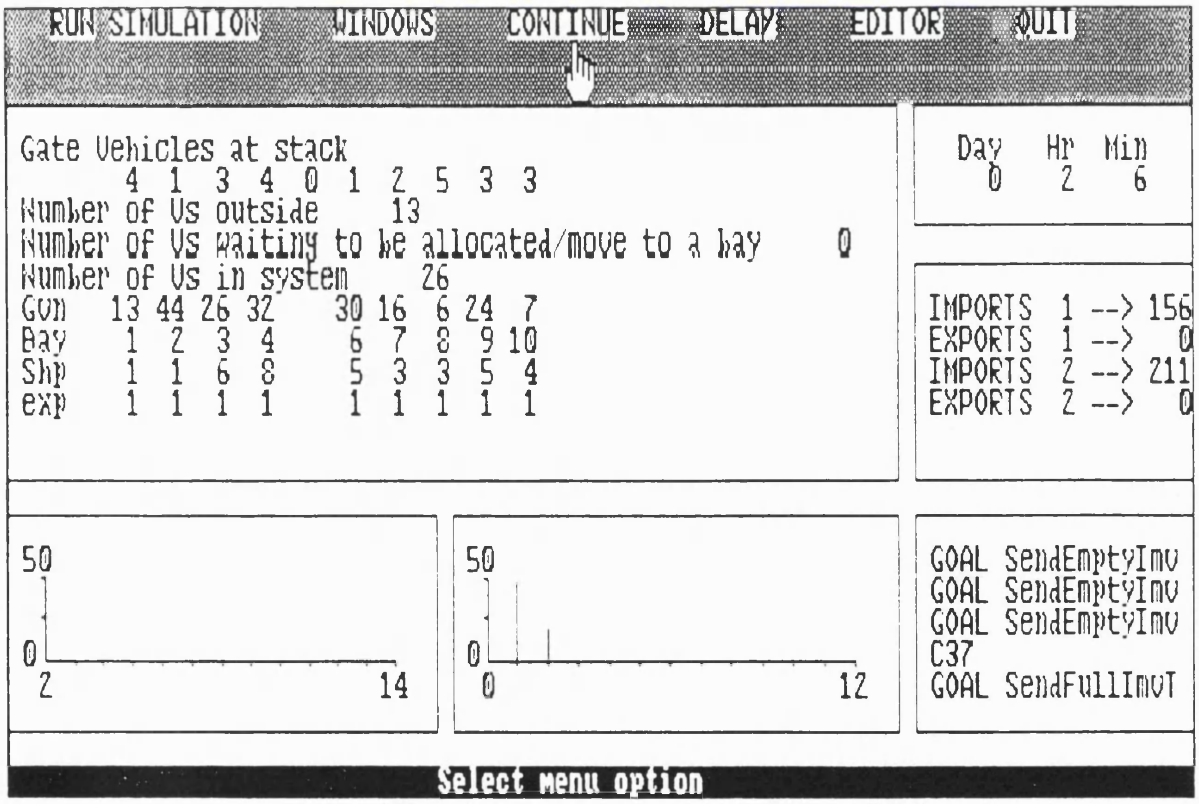
```
Opening File: IMVMANAG.TEM
Opening File: CRANEMAN.TEM
```

FIGURE G8.1 TRACE OF ESSIM COMPILATION OF RULE-SETS



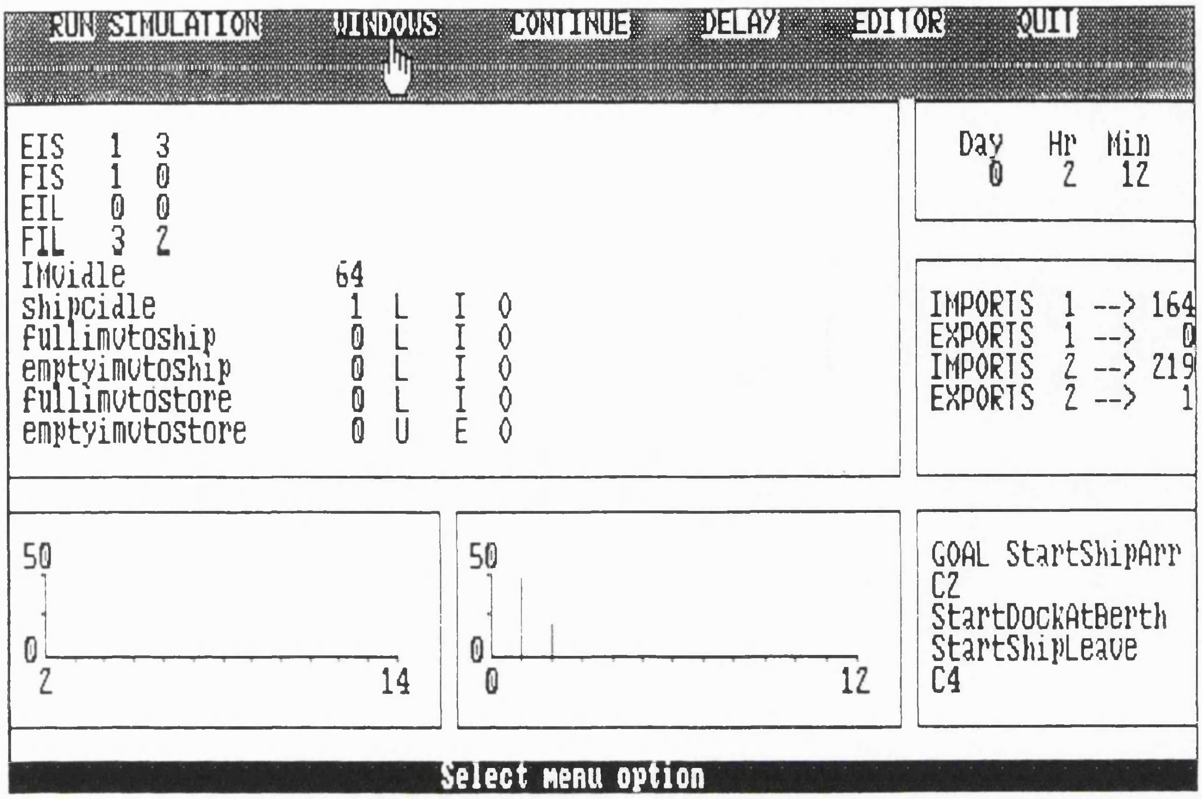
```
Delay (0..100) : ___

Berth1    Berth2
  2          1

I 500     I 500
E 500     E 500

Number Ships waiting to berth:   7

IMPORTS  1 --> 154
EXPORTS  1 -->   0
IMPORTS  2 --> 209
EXPORTS  2 -->   0

50          50
0           0
2     14    0     12

GOAL StartShipArr
C2
StartDockAtBerth
StartShipLeave
C4
```

FIGURE G82 TRACE DISPLAY OF SHIP CYCLE (POPUP 'DELAY OPTION ALSO SHOWN)



```
Gate Vehicles at stack
        4  1  3  4  0  1  2  5  3  3
Number of Vs outside      13
Number of Vs waiting to be allocated/move to a bay      0
Number of Vs in system      26
Gvn    13 44 26 32    30 16  6 24  7
Bay     1  2  3  4     6  7  8  9 10
Shp     1  1  6  8     5  3  3  5  4
exp     1  1  1  1     1  1  1  1  1

  Day   Hr   Min
   0     2    6

IMPORTS  1 --> 156
EXPORTS  1 -->   0
IMPORTS  2 --> 211
EXPORTS  2 -->   0

50          50
0           0
2     14    0     12

GOAL SendEmptyImv
GOAL SendEmptyImv
GOAL SendEmptyImv
C37
GOAL SendFullImvT
```

FIGURE G83 TRACE DISPLAY OF GATE VEHICLE CYCLE



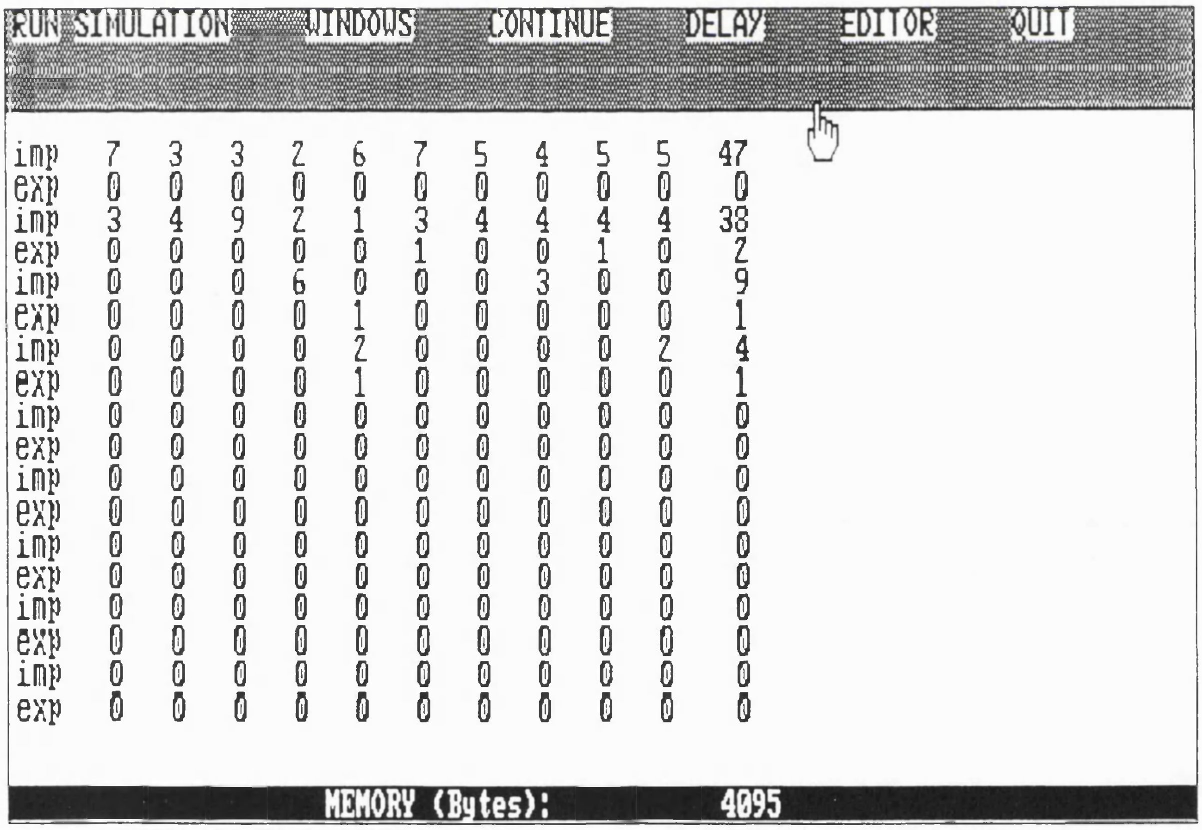
```
EIS   1  3
FIS   1  0
EIL   0  0
FIL   3  2
IMvidle                 64
shipcidle                1   L    I   0
fullimvtoship            0   L    I   0
emptyimvtoship           0   L    I   0
fullimvtostore           0   L    I   0
emptyimvtostore          0   U    E   0
```

```
Day   Hr   Min
 0     2    12
```

```
IMPORTS  1 --> 164
EXPORTS  1 -->   0
IMPORTS  2 --> 219
EXPORTS  2 -->   1
```

```
GOAL StartShipArr
C2
StartDockAtBerth
StartShipLeave
C4
```

FIGURE G84 TRACE DISPLAY OF IMV CYCLE



```
imp  7  3  3  2  6  7  5  4  5  5  47
exp  0  0  0  0  0  0  0  0  0  0   0
imp  3  4  9  2  1  3  4  4  4  4  38
exp  0  0  0  0  0  1  0  0  1  0   2
imp  0  0  0  6  0  0  0  3  0  0   9
exp  0  0  0  0  1  0  0  0  0  0   1
imp  0  0  0  0  2  0  0  0  0  2   4
exp  0  0  0  0  1  0  0  0  0  0   1
imp  0  0  0  0  0  0  0  0  0  0   0
exp  0  0  0  0  0  0  0  0  0  0   0
imp  0  0  0  0  0  0  0  0  0  0   0
exp  0  0  0  0  0  0  0  0  0  0   0
imp  0  0  0  0  0  0  0  0  0  0   0
exp  0  0  0  0  0  0  0  0  0  0   0
imp  0  0  0  0  0  0  0  0  0  0   0
exp  0  0  0  0  0  0  0  0  0  0   0
imp  0  0  0  0  0  0  0  0  0  0   0
exp  0  0  0  0  0  0  0  0  0  0   0

MEMORY (Bytes):        4095
```

FIGURE G85 TRACE DISPLAY SHOWING NUMBER OF CONTAINERS IN THE STACK AREAS

<u>APPENDIX H</u>

APPENDIX H

REFERENCES

Adelsberger,H.H. [1984], "Prolog As A Simulation Language", Proceedings Of The 1984 Winter Simulation Conference. British Computer Society.

Adelsberger,H.H. [1986], "Rule Based Object Orientated Simulation Systems", Proceedings Of The Conference On Intelligent Simulation Environments.

Alty,J.L. and Coombs,M.J. [1984], "Expert Systems - Concepts and Examples", NCC Publications, The National Computing Centre Ltd, Oxford Road, Manchester M1 7ED, England. ISBN 0-85012-399-2.

Alty,J.L. [1985], "The Limitations of Rule Based Expert Systems", in Knowledge-Based Expert Systems In Industry, Chapter 2.

Arumugam,V. [1985], "Priority Sequencing In A Real World Job Shop Simulation", Simulation 45:4, pp179-186.

Balci,O. [1986], "Requirements for Model Development Environments", Computers and Operations Research, 13:1.

Balci,O. and Nance,R.E. [1987], "Simulation Support: Prototyping. The Automation Based Paradigm", Technical Report 87-20, Department Of Computer Science. Virginia Tech. University.

Balmer,D.W., Goodman,D.H., and Doukidis,G.I., [1988], "Knowledge Based Management Support Systems.", Ed. Doukidis,G.I., Land,F., and Miller,G. Ellis Horwood Ltd, Chichester.

Balmer,D.W., and Paul,R.J. [1986], "CASM - The Right Environment For Simulation", Journal of the Operational Research Society, Vol. 37, N. 5, pp443-452.

Barrett,R.T. and Barman,S. [1986], "A SLAM II Simulation Study Of A Simplified Flow Shop", Simulation 47:5, pp181-189.

Baskaran,V. Fox,M. Sathi,N. and Bouer,J. [1986], "Simulation Craft: An AI Approach To Simulation Model Creation", Proceedings Of the IASTED Conference, June 4-6,1986, Vancouver, Canada.

Bassett,G and Kochar,A.K. [1985], "Decision Support For Material Requirements Planning Systems Using Computer Simulation", Proceedings Of The 1985 Summer Computer Simulation Conference, July 22-24,Chicago,Illinois, pp573-578.

Birtwistle,G.M. [1979], "Discrete Event Modelling on SIMULA", MacMillan, London.

Bitran,G.R., Hass,E.A. and Hax,A.C. [1982], "Hierarchical Production Planning: A Two-Stage System", Operations Research, 30:2, pp232-251.

Brooks,R. et al. [1979], "The Acronym Model Based Vision System", Proceedings Of The International Joint Conference On Artificial Intelligence. Volume 6, pp105-113.

Brown,T. Alexander,S. and Jagannathan,V. [1985], "Demonstration Of An Expert System For Manufacturing Process Control", AI, Graphics And Simulation, (Birtwistle,G. Ed), The Society For Computer Simulation, San Diego, California.

Buchanan,B.G. and Shortliffe,E.H. [1984], "Rule-Based Expert Systems: The MYCIN Experiments Of The Stanford Heuristics Programming Project", Addison-Wesley Publishing Company, Reading, MA.

Bullers,W.I. and Shultz,C.R. [1986], "Pprduction Rule-Based Simulation For Job Shop Scheduling", Proceedings of the Summer Simulation Conference. Reno, Nevada. pp718-723.

Chew,S.T. [1986], "Program Generators For Discrete Event Digital Simulation Modelling", Ph.d. Thesis, London School of Economics and Political Science.

Cleary,J. Goh,K. and Unger,B. [1985], "Discrete Event Simulation In Prolog", AI, Graphics & Simulation, The Society For Computer Simulation.

Clementson,A.T. [1982], "Extended Control And Simulation Language", CLE.COM Ltd, Birmingham."

Clementson,A.T. [1978], "Extended Control And Simulation Language/Computer Aided Programming System.", Reference Manual, University Of Birmingham, Birmingham, England.

Clocksin,W.F. and Mellish,C.S. [1984], "PROGRAMMING IN PROLOG", 2nd Edition, Springer-Verlag.

Costa,R.S. and Jardim,E.G.M. [1986], "Ouso da Simulacao Computacional no Plahejamento e Controle da Producao", Sao Paulo, 6th ANAIS, Sociedade Brasileira de Comando Numerio. SOBRACOM.

Crookes,J.G. and Valentine,B. [1982], "Simulation In Microcomputers", Journal of the Operational Research Society, N.33, pp855-858.

Doukidis,G.I. [1987], "An Anthology On The Homology Of Simulation With Artificial Intelligence", Journal Of The Operational Research Society Vol 38, N.8, pp701-712.

Doukidis,G.I. and Paul,R.J. [1991], "SIPDES", Expert Systems with Applications, Vol.2, No.2/3, pp 153-165.

Duda,R., Gaschnig,J. and Hart,P., [1979], "Model Design In The Prospector Consultant System For Mineral Exploration", Expert Systems In The Micro Electronic Age (D.Michie ed). Edinburgh University Press.

Erschler,J., Fontan.G. and Merce,M. [1986], "Consistency Of The Disaggregation Process In Hierarchical Planning", Operations Research, 34:3, pp464-469.

Feigenbaum,F.A. [1982], "Knowledge Engineering For The 1980's."
, Department Of Computer Science, Stanford University, Stanford, California.

Fishman,G.S. [1973], "Concepts And Methods In Discrete Event Digital Simulation", Wiley Press, New York.

Fjellheim,R.A. [1985], "A Knowledge Based Interface To Process Simulation", AI Applied To Simulation, Simulation Series, VOL.18 N.1.

Flitman,A.M. [1986], "Towards the Application of Artificial Intelligence Techniques for Discrete Event Simulation". Ph.d. Thesis, University of Warwick, School of Industrial and Business Studies.

Flitman,A.M. Hurrion,R.D. [1987], "Linking Discrete-Event Simulation Models With Expert Systems", Journal Of The Operational Research Society, Vol.38 N.8.

Fox,M.S. And Smith,S.F. [1984], "ISIS - A Knowledge Based System For Factory Scheduling", Expert Systems, Volume 1, No 1, pp25-47.

Futo,I. [1985], "Combined Discrete/Continuous Modeling And Problem Solving", AI, Graphics And Simulation. The Society For Computer Simulation.

Goodman,D.H. Balmer,D.W. and Doukidis,G.I. [1987], "Interfacing Expert Systems And Simulation For Job-Shop Production Scheduling", Proceedings Of The 3rd International Expert System Conference. Learned Information, Oxford. pp127-134, 2-4 JUNE 1987.

Gordon,G. [1981], "The Development Of The General Purpose Simulation System (GPSS)", History Of Programming Languages", (Wexelblat,R.L. Ed.), Academic Press, pp403-426.

Gosling,G.D. and Okseniuk,A.M. [1986], "SLICE - A System For Simulation Through A Set Of Cooperating Expert Systems", Applications Of AI In Engineering Problems, 1st International Conference, Southampton University.

Gray,P. and Borovits,I. [1986], "The Contrasting Roles Of Monte Carlo Simulation And Gaming In decision Support Systems", Simulation, 47:6, pp233-239.

Greig,I.D. [1979], "Validation, Statistical Testing And The Decision To Model", Simulation, 33:2.

Harmon,P. and King,D. [1985], "EXPERT SYSTEMS: Artificial Intelligence in Business", John Wiley & Sons, Inc.

Helman,D.H. and Bahuguna,A. [1986], "Explanation System For Simulation", Proceedings Of The 1986 Winter Simulation Conference.

Hill,T.R. and Roberts,S.D. [1987], "A Prototype Knowledge-Based Simulation Support System", Simulation 48:4 pp152-161.

Hills,P.R. [1971]. HOCUS P.E. Group, Egham, Surrey.

Hurrion,R.D. [1978], "An Investigation Of Visual Interactive Simumlation Methods Using The Job-Shop Scheduling Problem", Journal of the Operational Research Society, VOL.39, No.11, pp1085-1093.

IEEE [1984], "American National Standard for the Pascal Computer Programming Language", ISBN 0-471-88944-X, published by the IEEE, NY).

Jain,S. and Osterfeld,D. [1989], "Expert Simulation for On-Line Scheduling", 1989 Winter Simulation Conference, 4th-6th December 1989, The Capital Hilton Hotel, Washington D.C.

Kheir,N.A. and Holmes,W.M. [1978], "On Validating Simulation Models Of Missile Systems", Simulation, 30:4.

Kiran,A.S and Smith,M.L. [1983], "Simulation Studies In Job Shop Scheduling: A Survey", Ed. Haluk Bekiroglu, Simulation in Inventory and Production Control, Proceedings of the Conference on Simulation in Inventory and Production Control.

Klahr,P. [1984], "Artificial Intelligence Approaches To Simulation", Proceedings of the 1984 UKSC Conference On Computer Simulation, Published By Butterworth, Bath.

Klahr,P. [1985], "Expressibility In ROSS: An Object-Orientated Simulation System", AI Applied To Simulation, Simulation Series, VOL.18, N.1.

Klahr,P. and Faught,W.S. [1980], "Knowledge Based Simulation", Proceedings Of The First Annual National Conference On AI, California, pp181-183.

Koskossidis,D.A. and Davies.B. [1987],"Validation And Verification Of Job Shop Simulation Models"

Langen,P.A. [1985], "Application Of Artificial Intelligence Techniques To Simulation", Simulation and AI, Simulation Series, VOL.18, N.3.

Lavery,R.G. [1986], "Artificial Intelligence And Simulation: An Introduction", Proceedings Of The 1986 Winter Simulation Conference.

Markowitz,H.M. Hausner,B. and Karr,H. [1963], "SIMSCRIPT: A Simulation Programming Language", RAND Corporation RM-3310-PR, Prentice Hall, Englewood Cliffs, New Jersey.

Matthewson,S.C. [1975], "Interactive Simulation Program Generators", Proceedings Of The European Computing Conference On Interactive Systems, Brunel University.

McArthur,D.J., Klahr,P. Narain,S. [1986], "ROSS: An Object Orientated Language For Constructing Simulations", Expert Systems Techniques, Tools And Applications, PP70-94, Adisson Wesley, Reading, Masechussets.

McFall,M.E. and Klahr,P. [1986], "Simulation With Rules And Objects", Proceedings Of the 1986 Winter Simulation Conference.

McRoberts,M. Fox,M. and Hussain,N. [1986], "Generating Model Abstraction Scenarios In KBS", AI Graphics And Simulation. SCS, San Diego, California.

Meadows,R. [1988], "Simulation - You Can Fake The Real Thing", Port Development International.

Mellichamp,J. Kwon,O. and Wahab,A. [1987], "Desiging Flexible Manufacturing Systems With Expert System Technology", 1987 Summer Computer Simulation Conference.

Middleton,S. and Zanconato,R. [1985], "BLOBS: An Object Orientated Language For Simulation And Reasoning", Artificial Intelligence Applied To Simulation. pp130-135. The Society For Computer Simulation, VOL.18, N.1.

Mihran,G.A. [1972], "Some Practical Aspects Of The Verification And Validation Of Simulation Models", Operational Research Quartely, 23:1.

Miller,D.P. [1986], "Temporal Reasoning", Proceedings Of The 1986 Winter Simulation Conference.

Moser,J.G. [1986], "Integration Of AI And Simulation In A Comprehensive Decision-Support System", Simulation, 47:6, pp223-229.

Muetzelfeldt,R. Bundy,A. Uschold,M. and Robertson,D. [1985], "ECO - "An Intelligent Front End For Ecological Modelling", AI Applied To Simulation, Simulation Series, The Society For Computer Simulation, VOL.18, N.1.

Muller,C. [1986], "MODULA--PROLOG- A Programming Environment For Building Knowledge Systems", Knowledge-Based Expert Systems In Industry.

Muller,C. [1986], "MODULA-PROLOG: A Software Development Tool.", IEEE Software.

Nance,R.E. [1981], "The Time and State Relationships In Simulation Modelling", Communication Of The ACM, 24:4, pp173-179.

Nathan,D.L. and Sokol,D.Z. [1986], "A Decision Support Framework For Manufacturing Simulation Models", The Proceedings Of The 1986 Summer Simulation Conference, July 28-30, Reno, Nevada, pp737-740.

Naylor,T.M. and Finger,J.M. [1967], "Verification Of Computer Simulation Models", Management Science, 14:2.

Nestman,C.H. and Windsor,J.C. [1985], "Decision Support Systems: A Perspective For Industrial Enginners", IIE Transactions, 17:1, pp38-46.

O'Keefe,R. [1986], "Simulation And Expert Systems - A Taxonomy And Some Examples", Simulation, 46:1, pp10-16.

O'Keefe,R.M. [1986], "The 3-P Approach: A Comment On Strategy-Related Characteristics Of DIS Languages and Models", Simulation, 47:5, pp208-211.

O'Keefe,R.M. Balci,O. and Smith,E. [1986], "Validating Expert System Performance", IEEE Expert, Winter 1987, pp81-90.

O'Keefe,R.M. and Roach,J.W. [1987], "Artificial Intelligence Approaches To Simulation", Journal Of The Operational Research Society, Volume 38, N.8.

Overstreet,C.M. and Nance,R.E. [1985], "A Specification Language To Assist In Analysis Of Discrete Event Simulation Models", Communications Of The ACM. VOL.28 No.2.

Paul,R.J. and Doukidis,G.I. [1986], "Further Development In The Use of Artificial Intelligence Which formulate Simulation Problems", Journal of the Operational Research Society, No.37, pp787-810.

Pidd,M. [1992], "Computer Simulation In Management Science", 3rd Edition, John Wiley & Sons.

Prakash,S. and Shannon[1989], "Intelligent Back End of a Goal Directed Simulation Environment For Discrete-Part Manufacturing", 1989 Winter Simulation Conference, 4th-6th December 1989, The Capital Hilton Hotel, Washington D.C.

Pratt,W.K. [1978], "Digital Image Processing", J.Wiley & Sons, New York, 1978.

Reddy,R. [1987], "Epistomology Of Knowledge Based Simulation", Simulation, 48:8, pp162-166.

Reddy,Y.V. and Fox,M. [1986], "The Knowledge-Based Simulation System", IEEE Software, 3:2, pp26-37.

Reddy,Y.V., Fox,M.S., Doyle,K., Arnold,J. [1983], "INET: A Knowledge Based Simulation Model Of A Corporate Distribution System", Proceedings Of The IEEE Conference On Trends And Applications, Gaithersburg, MD.

Robertson,P. [1986], "A Rule Based Expert Simulation Environment", Proceedings Of The Conference On Intelligent Simulation Environments.

Rozenblit,J.W. and Zeigler,B.P. [1985], "Concepts For Knowledge-Based System Design Environments", Proceedings Of The 1985 Winter Simulation Conference (San Francisco, California). IEEE, pp223-231.

Ruiz-Mier,S. and Talavage,J. [1987], "A Hybrid Paradigm For Modeling Of Complex Systems", Simulation, 48:4, pp135-141.

Ruiz-Mier,S. Talavage,J. and Ben-Arieh,D. [1985], "Towards A Knowledge-Based Network Simulation Environment", Proccedings Of The 1985 Winter Simulation Conference, pp232-236.

Sathi,N. and Bauer,J. [1986] Simulation Craft: An AI Approach To The Simulation Life Cycle", Proceedings Of The 1986 Summer Computer Simulation Conference,July 28-30, Reno,Nevada. pp773-778.

Schlesinger [1974], "Developing Standard Procedures For Simulation Validation And Verification", Proceedings Of The 1974 Summer Computer Simulation Conference, VOL.1.

Schruben,L.W. [1980], "Establishing The Credibility Of Simulations", Simulation, 34:3.

Shannon,R.E. [1986], "Intelligent Simulation Environments", Proceedings Of The Conference On Intelligent Simulation Environments.

Shannon,R.E. Mayer,R. and Adelsberger,H. [1985], "Expert Systems And Simulation", Simulation, 44:6, pp275-284.

Shaw,M.L. and Gaines,B.R. [1986], "A Framework For Knowledge-Based Systems Unifying Expert Systems And Simulation", Proceedings Of The Conference On Intelligent Simulation Environments.

Shortliffe,E.H. [1976], "Computer-Based Medical Consultant: MYCIN", Elsevier, New York.

Stewart,D. and Surgenor,B. [1987], "Simulation Validation Of An Expert System For Process Fault Diagnosis", 1987 Summer Computer Simulation Conference pp663-667.

Talavage,J.J. [1978], "Models For The Automatic Factors", Simulation, 30:3.

Tocher,K.D. [1962], "The Art Of Simulation", English University Press .

Turing,A.M. [1950], "Computer Machinery And Intelligence", Mind, Volume 59, pp433-460.

Ulgen,O.M. and Thomasma,T. [1986], "Simulation Modeling In An Object-Oriented Environment Using Smalltalk-80", Proceedings Of The 1986 Winter Simulation Conference.

Van Horn,R.L. [1971], "Validation Of Simulation Results", Management Science, 17:5.

Vujosevic,R. [1990], "Object Oriented Visual Interactive Simulation", 1990 Winter Simulation Conference, 9th-12th December 1990, The Fairmont Hotel, New Orleans, Louisiana.

Winograd,T. [1972], "Understanding Natural Language", Academic Press, New York.

Zeigler,B.P. [1976], "Theory Of Modeling And Simulation", Wiley, New York.

APPENDIX I

APPENDIX I

BIBLIOGRAPHY

Adelsberger,H.H. and Neumann,G. [1985], "Goal Orientated Simulation Modeling Using Prolog", Modeling And Simulation On Microcomputers, pp42-47. San Diego, California

Aggarval,R. [1981], "A Simulation Model For Managing Foreign Exchange In A Multinational Company", Simulation In Business Planning And Decision Making (Naylor,T.H. Ed.). Simulation Proceedings Series. SCS. pp49-57.

Alter,S.L. [1980], "Decision Support Systems: Current Practice And Continuing Challenge", Addison-Wesley Publishing Co. Reading, Massechussets.

Ariav,G. and Ginzberg,M.J. [1985], "DSS Design: A Systemic View Of Decision Support", Communications Of The ACM, 28:10, pp1045-1052.

Arons,H. and De Swann [1983], "Expert Systems In The Simulation Domain", Mathematics And Computers In Simulation XXV, North Holland.

Baker,C.T. and Dzielinski,B.P. [1960], "Simulation Of A Simplified Job Shop", Management Science, 6:3, pp311-323.

Bell,R. and Bilalis,N.G. [1982], "Loading And Control Strategies For An FMS For Rotational Parts", Proceedings Of The First International Conference On FMS, Brighton, U.K.

Berry,W.L. [1972], "Priority Scheduling And Inventory control In Job Lot Manufacturing Systems", AIIE Transactions, 4:4, pp267-276.

Blackwell,R. [1986], "A Discrete Event Scheduler In A Dynamic Production System", Proceedings Of The 1986 Winter Simulation Conference.

Bratko,I. [1986], "PROLOG Programming For Artificial Intelligence", Addison-Wesley, Reading, Mass.

Broda,K. and Gregory,S. [1984], "PARLOG For Discrete Event Simulation", Research Report Document 84/4. Department OfF Computing, Imperial College Of Science & Technology, University Of London.

Brookes,C.H.P. [1985], "A Framework For DSS Development", Transactions Of The Fifth International Conference On Decision Support Systems. Institute Of Management Sciences.

Brown,R.G. [1968], "Simulation To Explore Alternative Sequencing Rules", Naval Research Logical Quartely, 15, pp281-286.

Brown,T Alexander,S. and Jagamathan,V. [1985], "Demonstration Of An Expert System For Manufacturing Process Control.", AI, Graphics And Simulation, The Society For Computer Simulation.

Browne,J. and Davies,B.J. [1984], "The Design & Validation Of A Digital Simulation Model For Job Shop Control", Internation Journal For Production Research, Vol.22, No.2, pp335-357.

Bulkin,M.H. Colley,J. and Steinhoff,H. [1966], "Load Forecasting, Priority Sequencing, and Simulation In A Job Shop Control System", Management Science, 13, B29-B51.

Bullers,W.I. Et. Al. [1980], "Artificial Intelligence In Manufacturing Planning and Control", AIIE Transactions, 12:4, pp.351-363.

Buxton,J.N. and Laski,J.G. [1962], "Control and Simulation Language", Computer Journal 5, pp194-199.

Calu,J. Et. Al. [1984], "Knowledge-Base Aspects in Advance Modelling and Simulation", Proceedings of the 1984 Summer Computer Conference.

Cash,C.R. and Wilhelm,W.E. [1986], "A Simulation Modeling Approach For Analysing Robotic Assembly Cells.", Proceedings of the 1986 Winter Simulation Conference.

Cheng,T.C.E. [1985], "Simulation of Flexible manufacturing Systems", Simulation 45:6, pp.299-302.

Colley,J.L. [1968], "Implementing A Job Shop Scheduling System", System Procedures Journal, 19, pp.28-33.

Conway,R.W. [1965], "Priority Dispatching and Work-in-Progress Inventory in a Job-Shop", Journal Of Industrial Engineering, 16:2, pp.123-130.

Conway,R.W. [1965], "Priority Dispatching and Job Lateness In a Job Shop", Journal Of Industrial Engineering, 16, pp.228-237.

Conway,R.W. Et. Al. [1967], "Theory Of Scheduling", Addison-Wesley.

Cox,J.F. and Adams,F.P. [1981], "Manufacturing Resource Planning: An Integrated Decision-Support System", Simulation In Business Planning And Decision Making. (Ed. Naylor,T.H.), Simulation Proceedings Series. SCS. pp.1-7.

Crookes,J.G. [1987], "Generators, Generic Models Ans Methodology.", Journal of the Operational Research Society, 38:8, pp.765-768.

Crookes,J.G., Balmer,D.W., Chew,S. and Paul,R.J. [1986], "A Three-Phase Simulation System Written In Pascal", Journal Of The Operational Research Society, 37, pp.603-618.

Davies,R. [1980], "Meta-Rules: Reasoning About Control", Artificial Intelligence, 15:3, pp.179-222.

Davis,D.A. [1986], "Modeling AGV Systems", Proceedings Of The 1986 Winter Simulation Conference.

Deliyanni,A. and Kowalski,R.A. [1979], "Logic and Semantic Networks.", Communications of the ACM, 22:3, pp.184-192.

Dogramaci,A. and Adam,N.R. [1979], "Current Issues In Computer Simulation.", Academic Press.

Eilon,S. and Cotterill,D.J. [1968], "Modified SI Rule On Job Shop Scheduling", International Journal Of Production Research, 7:2, pp.135-145.

Eilon,S. and Hodgson,R.M., [1967], "Job Shop Scheduling With Due Dates", International Journal Of Production Research, 6:1, pp.1-13.

Elmaghraby,S.E. and Cole,R.T. [1963], "On The Control Of Production In Small Job Shops", Journal Of Industrial Engineering, 14:4, pp.168-196.

Elvers,D.A. [1973], "Job Shop Dispatching Rules Using Various Delivery Date Setting Criteria.", Production And Inventory Management, 14:4,pp.62-80.

Emery,J.C. [1969], "Job Shop Scheduling By Means Of Simulation And An Optimum Seaking Search", Proceedings Of The Conference On Simulation, Los Angeles.

Evans,J.B. [1984], "Simulation, An Intelligence", Technical Report TR-A5-84, Centre Of Computer Studies And Applications, University Of Hong Kong.

Futo,I. Gergely,T and Deutsch,T. [1985], "Logic Modelling", AI Applied To Simulation, Simulation Series, The Society For Computer Simulation, 18:1.

Futo,I. and Szeredi,J. [1982], "A Discrete Simulation System Based On Artificial Intelligence Techniques", Discrete Simulation And Related Fields. (Javor,I. Ed.) pp.135-150. North-Holland.

Garzia,R.F. [1986], "Simulation With GPSS/PC", Proceedings Of The 1986 Winter Simulation Conference.

Gere,W.S. [1966], "Heuristics In Job Shop Scheduling", Management Science, 13:3.

Goldberg,A. and Robson,D. [1983], "Smalltalk-80: The Language And Its Implementation", Addison-Wesley, Reading, Massachusetts.

Goyal Et. Al. [1985], "COMPASS", Expert Systems, 2:3.

Henriksen,J.O. and Schriber,T.J. [1986], "Simplified Approaches To Modeling Accumulating And Nonaccumulating Conveyor Systems", Proceddings Of The 1986 Winter Simulation Conference.

Hershauer,J.C. and Ebert,R.J. [1975] ,"Search And Simulation Selection Of a Job-Shop Sequencing Rule", Management Science, 2:7.

Hollier,R.H. [1968], "A Simulation Study Of Sequencing In Batch Production", Operational Research Quartely, 19, pp.338-407.

Holloway,C.A. and Nelson,R.T. [1974], "Job Shop Scheduling With Due Dates And Overtime Capability", Management Science, 21:1.

Holloway,C.A. and Nelson,R.T. [1974], "Job Shop Scheduling With Due Dates And Variable Processing Times", Management Science, 20:9.

Holloway,C.A. and Nelson,R.T. [1973], "Alternative Formulation Of The Job Shop Problem With Due Dates", Management Science, 20:1.

Hottenstein,M.P. [1970], "Expediting In Job-order-Control Systems: A Simulation Study", AIIE Transactions, 2:1.

Hurrion,R.D. and Secker,R.J.R. [1978], "Visual Interactive Simulation: An AID To Decision Making", OMEGA 6, pp.419-426.

Jones,C.H. [1973], "An Economic Evaluation Of Job Shop Dispatching Rules", Management Science, 20, pp.293-307.

Jones,D.W. [1986], "Concurrent Simulation: An Alternative To Distributed Simulation", Proceedings Of The 1986 Winter Simulation Conference".

Kerckhoffs,E. Et. Al. [1985], "General Considerations On AI Applied To Simulation", AI Applied To Simulation, Simulation Series, 18:1.

Klahr,P. Ellis,J. Giarla,W. Narain,S. Cesar,E. and Turner,S. [1986], "TWIRL: Tactical Warfare In The ROSS Language.", Expert Systems: Techniques, Tools And Applications. pp.70-94. Addison-Wesley.

Klahr,P. McArthur,D. and Narain,S. [1982], "SWIRL: An Object-Oriented Air Battle Simulator.", Proceedings Of The 2ND Annual National Conference On Artificial Intelligence. Pittsburgh, pp.331-334.

Knapp,V. [1986], "The Smalltalk Simulation Environment.", Proceedings Of The 1986 Winter Simulation Conference.

Kowalski,R. [1979], "Algorithm = Logic + Control", Communications Of The ACM, 22:7, pp.424-436.

Kumar,D. [1986], "A Novel Approach To Sequential Simulation.", IEEE Software.

Legrande,E. [1963], "The Development Of A Factory Simulation Using Actual Operating Data", Management Technology, 8:1, pp.1-19.

Lirov,Y. Rodin,E. McElhaney,B. and Wilbur,L. [1988], "Artificial Intelligence Modelling Of Control Systems", Simulation, 50:1, pp.12-24., ISBN 0037-5497/88.

Mamalis,A.G. Bilalis,N. and Konstantinidis,M. [1987], "On Simulation Modeling For FMS", Simulation, 48:1, pp.19-23.

Martin,D.L. [1986], "Simulation Analysis Of An FMS During Implementation", Proceedings Of The 1986 Winter Simulation Conference.

Mayer,R., Young,R. and Mamalis,A.[1984], "An Assessment Of AI Applications To Manufacturing.", Industrial Automation Laboratory, Industrial Department, Texas A&M University.

Moore,J.M. and WisonN,R.G. [1967], "A Review Of Simulation Research In Job Shop Scheduling", Journal Of Production Inventory Management, 8, pp.1-10.

Moreira Da Silva,C. [1985], "The Use Of Decision Mechanisms In Visual Simulation For FMS Modelling", AI Applied To Simulation, Simulation Series, 18:1, The Society For Computer Simulation.

Nelson,R.T. [1967], "Labor And Machine Limited Production Systems", Management Science, 13:9.

Nof,S.Y. Whinston,A. and Bullers,W. [1980], "Control And Decision Support In Automatic Manufacturing Systems", AIIE Transactions, 12:2, pp.156-169.

Nolan,P.J. and McCarthy,M.A. [1986], "AI Frame-Based Simulation In System Dynamics.", Applications Of AI In Engineering Problems, 1st International Conference, Southampton University.

Norman,T.A. and Norman,V.B. [1986], "Interactive Factory Scheduling Using Discrete Event Simulation.", Proceedings Of The 1986 Winter Simulation Conference.

O'Keefe,R.M. [1986], "Experiences With Using Expert Systems In OR.", Journal Of The Operational Research Society, Number 37, pp.657-668.

O'Keefe,R.M. [1985], "Expert Systems And Operational Research - Mutual Benefits.", Journal Of The Operational Research Society, Number 36, pp.125-130.

Orciuch,E. and Frost,J. [1984], "ISA: Intelligent Scheduling Assistant", IEEE.

Oren,T.I. [1986], "Knowledge Bases For An Advanced Simulation Environment.", Proceedings Of The Conference On Intelligent Simulation Environments.

Oren,T.I. [1985], "Artificial Intelligence and Simulation.", AI Applied To Simulation, Simulation Series, 18:1.

Oren,T.I. and Zeigler,B.P. [1979], "Concepts For Advanced Simulation Methodologies.", Simulation, 32:3, pp.69-82.

Overstreet,C.M. Nance,R. Balci,O. and Barger,L. [1986], "Specification Languages:Understanding Their Role In Simulation Model Development.", Technical Report SRC-87-001, Department Of Computer Science, Virginia Tech. University.

Panwalkar,S.S. and Iskander.W. [1977], "A Survey Of Scheduling Rules.", Operations Research, 25:1.

Radzikowski,P. [1983], "Perspectives Of The Business Decision Support Expert System.", TIMS/ORSA.

Raghunath,S. and Perry,R. and Cullinance,T. [1986], "Interactive Simulation Modelling Of Automated Storage Retrieval Systems", Proceedings Of The 1986 Winter Simulation Conference.

Rochette,R. and Sadowski,R.P. [1976], "Statistical Comparison Of The Performance Of Simple Dispatching Rules", International Journal Of Production Research, 14:1.

Rossley,T.R. [1983], "Simulation Of A FMS For The Manufacture Of Sheet Metal Components", Annals Of The CIRP, 32:1, pp.427-431.

Rothenberg,J. [1986], "Object-Oriented Simulation: Where Do We Go From Here?", Proceedings Of The 1986 Winter Simulation Conference.

Sargent,R.G. [1986], "Joining Existing Simulation Programs.", ProCeedings Of The 1986 Winter Simulation Conference.

Sprague,R.H. and Carlson,E.D. [1982], "Building Effective Decision Support Systems", Prentice-Hall Inc., Englewood Cliffs, N.J.

Stecke,K.E. and Solberg,J.J. [1981], "Loading And Control Policies For A Flexible Manufacturing System", International Journal Of Production Research, 19:5, pp.481-490.

Subrahmanyam,P.A. [1985], "The Software Engineering Of Expert Systems: Is Prolog Appropriate?", IEEE Transactions, Software Engineering, Vol. SE-11, No. 11, pp.370-386.

Symankiewizk,J., McDonald,J. and Turner,K. [1988], "Solving Business Problems by Simulation", 2nd Edition, Mc Graw-Hill, London.

Unger,B.W. [1986], "Object Oriented Simulation - ADA, C++, SIMULA", Proceedings Of The 1986 Winter Simulation Conference.

Vere,S.A. [1983], "Planning In Time: Windows And Durations For Activities And Goals.", IEEE Transactions On Pattern Analysis And Machine Intelligence, 5:3, pp.246-267.

Wilbrecht,J.K. and Prescott,W. [1969], "The Influence Of Setup Time On Job Shop Performance", Management Science, 16:4.

Wright,M. Et. Al. [1986], "An Expert System For Real-Time Control.", IEEE Software, March 1986.

Yan,J.C. and Lundstrom,S.F. [1986], ""AXE": A Simulation Environment For Actor-Like Computations On Ensemble Architectures.", Proceedings Of The 1986 Winter Simulation Conference.

Young,R.E. and Meyer,R. [1984], "History And Introduction To Artificial Intelligence And Expert Systems.", Working Paper, Industrial Automation Laboratory, Industrial Engineering Department, Texas A&M University.