

x-21-158571-7
F6953



SEMANTIC MODELLING
FOR
DISCRETE EVENT SIMULATION

MAMDOUH TAYSIR BARAKAT

The London School of Economics

Thesis submitted in fulfilment of the requirement
for the degree of Doctor of Philosophy at
the London School of Economics,
University of London.

1992

UMI Number: U615772

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



UMI U615772

Published by ProQuest LLC 2014. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.

All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.



ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

ABSTRACT

Discrete event simulation modelling has been established as an important tool for management planning. This process has been aided by the availability of off-the-shelf simulation systems for microcomputers. Traditionally these have had text-based interfaces and very limited graphics. As the availability of powerful colour microcomputers have increased, graphical front-ends have been added. As clients have got used to consistent graphical interfaces (e.g. Apple Macintosh or Microsoft Windows), they have desired the same level of integration in their simulation support environments.

Research in other fields has been utilised in improving simulation environments. These fields include relational databases, expert systems, formal languages and graphical environments. This thesis examines the use of artificial intelligence in the discrete event simulation field with the aim of examining some potential areas in which it might be possible to improve simulation environments.

Existing simulation research in the artificial intelligence (AI) field is extended by investigating the graphical AI knowledge-base called semantic networks. This thesis demonstrates semantic modelling, a discrete event simulation modelling approach based on semantic networks, which attempts to give a consistent graphical interface throughout the life cycle of a simulation study. The semantic modelling approach also incorporates expert system and natural language research. A prototype system of this approach is described.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr Ray J. Paul for all the help and support he has provided.

I would also like to thank my family and members of the LSE's Department of Information Systems for their encouragement throughout this research.

CONTENTS

CHAPTER 1 : INTRODUCTION

1.1 : INTRODUCTION	1
1.2 : SIMULATION MODELLING	2
1.3 : DISCRETE EVENT SIMULATION	2
1.4 : TRADITIONAL SIMULATION DEVELOPMENT CYCLE	4
1.5 : ARTIFICIAL INTELLIGENCE	11
1.6 : THESIS OUTLINE AND OBJECTIVES	13

CHAPTER 2 : AI LITERATURE IN SIMULATION

2.1 : INTRODUCTION	14
2.2 : HISTORICAL LINK BETWEEN SIMULATION AND AI	15
2.2.1 : Production rules in simulation	15
2.2.2 : Expert systems in simulation	18
2.2.3 : AI concepts in simulation	19
2.3 : INTEGRATED SIMULATION SUPPORT ENVIRONMENTS	23
2.3.1 : The goal	24
2.3.2 : Activity cycle diagram approach	25
2.3.3 : Database approach	27
2.3.4 : Object-orientated language approach	29
2.3.5 : Expert Systems (production rules) approach	30
2.3.6 : Semantic network approach	31
2.4 : CONCLUSION	33

CHAPTER 3 : SEMANTIC MODELLING : DESIGN ISSUES

3.1 : INTRODUCTION	36
3.2 : UNDERLYING STRUCTURE OF SEMANTIC NETWORKS	36
3.2.1 : Synonyms	38
3.2.2 : Core objects, instances and the "IS A" relationship	39
3.2.3 : The "PRECEDES" relationship	40
3.2.4 : The "DURATION" relationship	41

3.2.5 : The "NUMBER" and "INIT NUMBER" relationship	42
3.2.6 : Examples	43
3.3 : GRAPHICAL INTERFACE	44
3.4 : NATURAL LANGUAGE UNDERSTANDING AND PROCESSING	47
3.4.1 : Syntactic Analysis during NLUP	48
3.4.2 : Semantic Analysis during NLUP	51
3.5 : ACTIVITY AND ENTITY CYCLE INPUT	53
3.6 : PROCEDURAL ATTACHMENT OF PRODUCTION RULES	53
3.7 : VALIDATION AND VERIFICATION	54
3.8 : INFERENCE ENGINE	56
3.9 : SIMULATION RUNNING	61
3.10: SEMANTIC MODELLING ARCHITECTURE	62
3.11: CONCLUSION	63

CHAPTER 4 : SASIM PROTOTYPE : IMPLEMENTATION ISSUES

4.1 : INTRODUCTION	68
4.2 : GRAPHICAL INTERFACE	69
4.3 : MULTI LEVEL MENU SYSTEM	74
4.3.1 : Worksheet Sub-menu	75
4.3.2 : File Sub-menu	76
4.3.3 : List Sub-menu	77
4.3.4 : Move Sub-menu	78
4.3.5 : Delete Sub-menu	78
4.3.6 : Hide Sub-menu	78
4.3.7 : Infer Sub-menu	78
4.3.8 : Validate Option	80
4.3.9 : Simulation Run Option	81
4.3.10 : Quit Sub-menu	81
4.4 : NATURAL LANGUAGE UNDERSTANDING AND PROCESSING (NLUP)	81
4.5 : ACTIVITY AND ENTITY CYCLE INPUT INTERFACE	83
4.6 : SIMULATION RUNNING PROCESS	86
4.7 : CONCLUSION	87

CHAPTER 5 : PROCEDURAL ATTACHMENT	
5.1 : INTRODUCTION	88
5.2 : STANDARD TERMINOLOGY	90
5.3 : STRUCTURE OF PASCAL USER CODE	91
5.4 : CALLABLE PROCEDURES	98
5.5 : COMPILING USER CODE	104
5.6 : AUTO GENERATING USER CODE	104
5.7 : CONCLUSION	105
CHAPTER 6 : ANALYSIS OF SEMANTIC MODELLING	
6.1 : INTRODUCTION	106
6.2 : THE PUB EXAMPLE	107
6.3 : WAR ARSENAL PROBLEM	110
6.4 : ANALYSIS OF SEMANTIC MODELLING	112
6.5 : CONCLUSION	114
CHAPTER 7 : SUMMARY AND CONCLUSION	
7.1 : SUMMARY	115
7.2 : CONCLUSION	116
7.3 : FUTURE RESEARCH	117
APPENDIX 1 : NATURAL LANGUAGE SYNTAX DEFINITION	120
APPENDIX 2 : NATURAL LANGUAGE CONVERSION MASKS	126
APPENDIX 3 : TEST OF SASIM ON THE PUB	
3.1 : NLUP TRANSLATION	129
3.2 : LIST OF LINKS	134
3.3 : LIST OF OBJECTS	136
3.4 : LIST OF RELATIONS	137
3.5 : OUTPUT FROM VALIDATE OPTION	138
3.5.1 : PROBABILITY SECTION	138

3.5.2 : ENTITY CYCLE SUMMARY	138
3.5.3 : ACTIVITY SUMMARY	140
3.6 : PASCAL MODIFICATION TO THE PUB EXAMPLE	141
3.7 : SCREEN PRINTOUTS DURING SIMULATION	145
APPENDIX 4 : WAR ARSENAL PROBLEM	159
APPENDIX 5 : TECHNICAL DATA AND INSTALLATION GUIDE	166
APPENDIX 6 : KNOWLEDGE REPRESENTATION IN MEMORY	167
APPENDIX 7 : GLOSSARY OF ABBREVIATIONS / NAMES	169
APPENDIX 8 : BIBLIOGRAPHY	172

CHAPTER 1

INTRODUCTION

Section 1.1 : INTRODUCTION

Discrete Event Simulation has been widely used for decades to analyse systems which could not themselves be experimented on in the real world, for either technical, social or financial reasons. It has proven itself in many real life situations as a very useful tool for the management and control of complex environments. This process has been considerably aided by the availability of inexpensive microcomputers. Recently, the availability of colour graphical environments (for example the Apple Macintosh computer or the Windows environment) have provided an impetus to increase the user friendliness of the text-based simulation systems. This thesis investigates whether semantic networks, an artificial intelligence knowledge-base, can be utilised in improving simulation environments.

The next two sections define both general simulation and discrete event simulation. The traditional simulation development cycle is described in section 1.4, which also highlights the lack of integration between the different stages of a simulation cycle. Section 1.5 introduces artificial intelligence and explains this thesis's interest in its tools and techniques. The thesis objective is explained in the final section.

Section 1.2 : SIMULATION MODELLING

One definition of simulation modelling is :

"Simulation involves experimentation on a model of some system. The model is used as a vehicle for experimentation, often in a 'trial and error' way to demonstrate the likely effects of various policies. Thus those which produce the best results would be implemented in the real system." Pidd (1988)

Computer simulation, where a simulation is performed with the aid of a computer, can be a very time consuming and complex task, despite the special purpose simulation languages, program generators and commercial packages available. This is mainly due to the time it takes to describe a problem to a computer simulation system, to test the system, to experiment with the system and finally to draw a conclusion from these experimentation. It is thus possible to regard computer simulation as a last resort, to be used if no other alternative is successful.

Section 1.3 : DISCRETE EVENT SIMULATION

Discrete event simulation is defined [Pidd, 1988] as a simulation that uses the "next event" time mechanism. This is a simulation where the time jumps from the present time to the time of the next event (ie the start or end of an activity). Only at this new time will the state of the system be examined and updated. This can only

be achieved if it is possible to 'predict' the earliest time of the next change of state. This can be achieved if the time it takes to actually change state can be sampled at the 'start of the change of state', and, once sampled, can not be changed. For example, the time it takes to drink a glass of beer can be sampled from a normal distribution whenever a customer begins to drink from the glass.

This approach is suitable where the variables (people, machines, goods) move from one distinct state to another distinct state (for example from one queue to another queue). This has the advantage of examining the system more frequently in times of high activity and less frequently in periods of low activity, considerably speeding up the simulation. Discrete event simulation also has the advantage of indicating clearly periods of high activity (time would move forward slowly) and vice-versa. However extra information must be stored for discrete event simulation (ie the timing tree), as opposed to a time-slicing approach where time jumps by a pre-determined fixed interval.

Section 1.4 : TRADITIONAL SIMULATION DEVELOPMENT CYCLE

The traditional process of simulation can be summarized, at the cost of over-simplification, to the four stages :



Stages	(i)	Problem formulation
	(ii)	Program generation
	(iii)	Simulation running (and rerunning)
	(iv)	Output analysis

Stage (i) is the problem formulation stage. It is a very difficult area to tackle in real life, and the techniques are not easily taught due to the multiple disciplines necessary (politics, economics, social sciences, computing, operational research, artificial intelligence). It is in this environment that early investigation into Natural Language Understanding and Processing (NLUP) had been undertaken, including SPIF [Doukidis, 1985] at the London School of Economics. However SPIF was conducted on the premise that the computer controlled the conversation with the client, who could only input his knowledge in direct response to a computer question. This was interfaced to the next stage of the development cycle.

The conceptual model is the output from the problem formulation

stage. It would be desirable if the conceptual model is unambiguous (as contrasted with standard spoken languages) in order to be able to be used as a direct input into an automatic program generator, as will be highlighted in stage (ii) below. It would also be desirable to be able to directly inspect and amend the conceptual model, since this would provide an immediate feed-back to the user, thus increasing the possibility of spotting and correcting errors at this stage. It would be advantageous if the client could be involved at this stage to increase the simulation's accuracy and increase the client's confidence in the model. Therefore flexibility, simplicity and transparency are desired characteristics for a conceptual model. However they could be contradictory (increased flexibility may lead to increased complexity and reduced transparency).

There are a wide variety of methods for representing the conceptual model of a problem [Ceric and Paul, 1989]. One of the most widely used conceptual models is an Activity Cycle Diagram (ACD), popularized by Clementson (1982), Hill (1971), Mathewson (1974), Pidd (1988), Syzmankiewicz et al (1988) and Tocher (1963). Its popularity is reflected in its use by a number of successful commercial simulation software packages, including HOCUS [Syzmankiewicz, 1984], CAPS program generator [Clementson, 1982], VS7 [Chapman and Dayer-Smith, 1990] as well as the research work of Au and Paul (1990).

Stage (ii) is the program generation stage. This is where the conceptual model is transformed (preferably automatically, for speed and accuracy) into a form that can be executed by a computer (for example from an ACD to pascal code). In some environments, including HOCUS [Syzmankiewicz, 1984] and VS7 [Chapman and Dayer-Smith, 1990], there is no requirement for a program generation stage, since a data-driven simulation could be run directly from the conceptual model. However, most simulations are too complex to be handled 100% by either directly running from the conceptual model or by using an automatic program generator. So a strategy that has been successfully used is the generation of a large portion of the code (around 70% [Crookes, 1987]) by an automatic program generator, and then let the simulation practitioner manually program the remainder. Powerful environments would enable the user to run a simple data-driven simulation, before generation of the code, to help the validation and verification of the model at an early stage, and then to automatically generate the bulk of the code. The programs generated need not be restricted to any specific modelling approach, including those listed by Pidd (1988) (process interaction approach, event-based approach, three phase approach etc) [Mathewson, 1974]. The concepts of program generation are similar to those used in the Programming by Questionnaire System of RAND [Oldfather et al, 1966]. There are now many Interactive Simulation Program Generators (ISPGs), notably CAPS [Clementson, 1982], DRAFT [Mathewson, 1977] and VS7 [Chapman and Dayer-Smith, 1990]. The

first and most famous [Paul and Chew, 1987] ISPG is CAPS [Clementson, 1982] which produces ECSL code from an ACD input. DRAFT can produce models coded in FORTRAN [Mathewson, 1977] [Mathewson, 1982] or in SIMULA [Mathewson and Beasley, 1976]. The use of program generators, with some manual programming by the simulation practitioner, can produce a very wide class of models, but forces the client to require "repeated recourse to the specialist as changes of model structure are required" [Crookes, 1987] since the client would generally not have enough computer knowledge to alter the simulation program himself. Crookes proposes an alternative approach of requiring the simulation practitioner to put in a big programming effort at the start in order to build a generic model, specific to a class of model which the client is interested in, and to build a special-purpose editor(s) which the client can use by himself, without program input, to modify the details of the generic model. However, since the generic model must be able to handle all particular instances of the chosen class of model (which even the client may not have thought of) it requires increased programming skills from the simulation practitioner and would take a longer time initially for the simulation practitioner to build his first working simulation than with a program generator approach. Naturally, if the simulation practitioner had already built the editor and generic model for another client, the simulation practitioner can just give the new client a copy of the editor and generic model and let him use it immediately, thus saving

time and effort.

The simulation program needs to be verified. Verification is the process of checking that the computer program corresponds to the conceptual model. Similarly, the conceptual model needs to be validated (the process of checking that the conceptual model corresponds to the real life process). These can be done by using existing data about the system and checking that the computer model behaves the same way as the real world system. If the system does not exist, this would require a thorough walk through of the code and sensitivity analysis to ensure that the model behaves as the client would anticipate. Without validation and verification the simulation program would give wrong results, and lead to wrong conclusions.

Stage (iii) is the simulation running (and rerunning) stage of the program. It needs, for efficiency, a good software subsystem. There are many such systems (Witness, Simscript II.5 [Caci, 1976], Modsim II etc) which increasingly use visual displays. Research in visual interactive modelling (VIM), which use a graphical display to both display the model and allow the user to interactive modify it (thus increasing the very important feedback to the user), include [Au and Paul, 1990] [Chapman and Dayer-Smith, 1990] [Hurrion, 1986] [Withers and Hurrion, 1982].

Stage (iv) is the output analysis stage. It is a relatively immature field, as it relies on common sense, judgement and inside knowledge of the real world system. This process, for general simulation, is well beyond the current capabilities of artificial intelligence (AI). Limited attempts have been made to provide an automatic linkage of the data from a simulation run to a statistical analysis package, such as Minitab or SPSS. One of these research attempts was done by Taylor and Hurrion (1988).

These stages are iterative. For example, it is usual to move from any of the stages (ii) - (iv) back to stage (i), the problem formulation stage, after discovering an anomaly in the computer model. It is therefore desirable not to lose any of the information / customisation added in the later stages. However this requirement for quick iteration is where the present simulation systems have difficulties. For example, the SPIF problem formulator creates a conceptual model. This conceptual model is not complete and would require some extra manually added data. Rerunning SPIF would result in the loss of the added data. This same problem affects the program generators, since once the simulation code is generated and then customised, rerunning the program generator would overwrite the customised code. Thus, the very distinct interfaces between the stages produce an implied single-direction chain of events, as opposed to the iterative concepts of simulation.

These different solutions, despite limited integration, unfortunately do not present a consistent interface to the simulation practitioner. As noted by Balci and Nance (1987), "Automated support of a simulation study throughout its entire life cycle is undeniably needed to confront the problems identified by Balci (1986)". As an example of the problems encountered, the graphical interface tends to be added after the model has already been developed. This, as noted by Paul (1988), is curious since the graphical output represents the computer model which represents the logical model which represents the real world problem, therefore why is the real world problem not expressed in graphical form first? Hurrion (1986) has noted the importance of the ability, during visual interactive modelling (VIM) with the user, to be able to respecify the model, rather than just limit the interaction to parameter changes [Withers and Hurrion, 1982] [O'Keefe, 1984]. Even though the extension of the visual interaction to the problem formulation stage has already been attempted by Interactive Simulation Program Generators (see above), these tend to have a different interface to the run-time VIM. If they do have a different interface, this would at the very least increase the learning time for users (since they would need to learn two interfaces, as opposed to one interface) and could potentially confuse the user.

There have been a number of attempts to integrate these stages into a single simulation support environment, for example SMDE [Balci and Nance, 1987] (based around a relational database), KBS [Baskaran and Reddy, 1984] (based around a schema representation language [Reddy and Fox, 1982]), ROSS [McArthur et al, 1986] (based around an English-like, interactive object-oriented language implemented in Lisp), JADE [Unger et al, 1986] and ANDES [Birtwistle et al, 1984]. These will be investigated in section 2.3 "Integrated Simulation Support Environments".

Section 1.5 : ARTIFICIAL INTELLIGENCE

Artificial intelligence (AI) is a broad term used to describe the research area for making machines, including computers, more human-like. This might result in both a better understanding of the nature of the human mind, and in making computers easier to use.

There are a number of separate research areas in AI, including Natural Language Processing, Expert Systems and Robotics. A large number of tools and techniques have been developed, for example object orientated computer languages, natural language interfaces and expert system techniques. A strong undercurrent in AI, also found in so-called 4th generation relational databases, is the desire for the explicit representation of data. The data for a situation/problem would thus be separated from the actual computer

code to create knowledge-bases. These knowledge-bases could be either textual (e.g. a list of objects), or graphical (displayable and modifiable as an image on a computer screen, such as the semantic network knowledge-base).

Chapter 2 will highlight the long-standing use of AI in simulation. It is worth noting at this stage that simulation practitioners have been using some of these techniques long before the AI field had investigated and labelled them as AI techniques. Recent simulation systems have used AI to aid their simulation environment. These systems will be examined in the next chapter. Nearly all these systems are at the research level [Paul, 1991]. Although Paul contends that there is no real application of a 'true' artificial intelligence system combined with simulation, I hope that this is a reflection on the short comings of the existing systems (or their marketing) rather than a more fundamental problem.

Section 1.6 : THESIS OUTLINE AND OBJECTIVES

This research project examines the use of artificial intelligence in the discrete event simulation field with the aim of examining some potential areas in which it might be possible to improve simulation environments. This thesis describes the results of this research.

Some of the current research in the use of artificial intelligence in simulation is described in the next chapter. The chapter describes the importance of the search for a suitable integrated simulation support environment. A number of AI and relational database techniques have previously been applied in this area, with varied practical success due to the environments' intrinsic complexity. The critical area which is examined is the knowledge representation (ie the knowledge-base), since this will ultimately decide the power, flexibility and user-friendliness of the resultant systems. As there already exists a graphical AI knowledge-base called semantic networks, it was a natural candidate to be investigated to see whether it can be utilised in improving simulation environments. This led to the development during this research of semantic modelling, a discrete event simulation modelling approach based on semantic networks. Semantic modelling is described in chapter 3. A prototype implementation of this approach is described in chapters 4 and 5. Chapter 6 presents an analysis of semantic modelling. Chapter 7 presents the summary and conclusion.

CHAPTER 2

AI LITERATURE IN SIMULATION

Section 2.1 : INTRODUCTION

This chapter highlights the long-standing uses of artificial intelligence (AI) in simulation. Even though this has resulted in benefits to both fields, this thesis concentrates on the benefits to simulation. Section 2.2 indicates that there are many elements in traditional simulation which are now regarded as artificial intelligence. Section 2.3 highlights the present international research in integrated simulation support environments. These environments are built around a knowledge-base which can be derived either from traditional simulation techniques (for example activity cycle diagrams), fourth generation techniques (databases) or from the artificial intelligence field (for example object-orientated languages).

This chapter will aim to provide the research foundation for this thesis by examining what a simulation environment should do and which aspects of both artificial intelligence and non-AI tools and techniques are probably appropriate to achieve these aims.

Section 2.2 : HISTORICAL LINK BETWEEN SIMULATION AND AI

Simulation and artificial intelligence (AI) are closely related in four ways [Doukidis, 1987] : methodological similarities, expert systems in simulation, AI concept usage in simulation, and the Gains for AI when applying the ideas from simulation (this last similarity is not considered further). These common areas are also examined by Paul (1989a, 1989b).

The next three sub-sections highlight each of these main similarities. The methodological similarities are described in Section 2.2.1 "Production Rules in Simulation". Section 2.2.2 describes expert systems usage in simulation. Section 2.2.3 describes the usage of AI concepts (specifically Natural Language Understanding and Processing and the object-orientated approach) in simulation.

Section 2.2.1 : PRODUCTION RULES IN SIMULATION

It is important to realise that production rules are not synonymous with expert systems, but they are one way of developing expert systems that represent knowledge explicitly [Browston et al, 1985] or non-explicitly [Doukidis, 1987]. Production rules were used in several fields well before being taken up by expert systems; in symbolic logic by Post (1943) and in linguistics by Chomsky (1957).

Production rules are the classic IF-THEN rules in the form :

```
IF    {condition}
THEN  {action}
```

They have naturally been used in computing programs to hold the branching conditions of programs, as championed by Tocher (1963) and Newell and Simon (1972), while in simulation programs they are used to hold the conditions required to start activities, as well as to decide on the route of entities [Vaucher, 1985] [O'Keefe, 1986a]. In this form they are called 'ACTIVE' rules, because data and the program are intercombined. Doukidis (1987) goes one step further, and says that simulation models are production-system models. An example of an active production rule might be :

```
VAR
    DOG, POSTMAN : ENTITY (* DECLARATION *)
PROCEDURE C_ACTIVITY
BEGIN
    IF ((QSIZE(DOG) >= 1) AND (QSIZE(POSTMAN) >= 1) THEN
        BEGIN
            {START BARK}
            {SCHEDULE END OF BARK IN x MINUTES}
        END; (* BARK occurs before BITE)
END;
```

QSIZE() is a function indicating the number of an entity waiting in a queue. As can be seen by the above example, we have explicitly defined the Dog and the Postman as entities at the start of the program. These must be declared by the simulation practitioner before their use, with activities being defined implicitly in the C_EVENT procedure of the simulation.

However it is apparent that to describe even a small simulation requires a large number of production rules. This could take a long time to enter. To solve this problem many interactive simulation program generators have been produced, either stand alone, such as DRAFT [Mathewson, 1977], or as part of a powerful simulation environment such as VS7 [Chapman and Dayer-Smith, 1990]. Thus the sole use of production rules, when describing a simulation, does not lend itself to the novice user, or even the experienced user, since the lack of transparency of the code can and does result in logical errors in coding which are very difficult to pinpoint (ie both validation errors in interpreting the client's description of the situation, and verification errors in interpreting the conceptual model). Program generators can help to both reduce the time it takes to describe a simulation, as well as reduce the chances of input errors, but only up to a point. And in any case, the eventual clients needs to be convinced of the accuracy of the model. These problems have been tackled by adding graphical interfaces onto the simulation, but unfortunately usually as an after-thought [Paul and Chew, 1987], with considerable extra effort by the practitioner.

It is worth noting that with the pressures from commercial expert systems, whose knowledge-base can be changed quickly and easily, active rules are being overshadowed in AI circles by textual (or 'PASSIVE' rules) [Flitman and Hurrion, 1987]. The advantage of passive production rules are that fundamental details in a

simulation can be changed during a simulation without needing to recompile the simulation program.

Section 2.2.2 : EXPERT SYSTEMS IN SIMULATION

Expert systems can be used to aid the user in developing his simulation model, whether being integrated into a simulation tool, such as the ruled-based program generator of Khoshnevis and Chen (1986), or stand-alone to help a simulation practitioner in both validation and verification.

An example of a stand-alone expert system is SIPDES [Doukidis and Paul, 1991], which is short for a "Simulation Program Debugger using Expert Systems". This aids in the debugging of a faulty simulation program written for the eLSE environment [Crookes et al, 1986]. The problems covered are both run-time errors and logical errors. The theory is that this will reduce the burden on the few experts who knew the eLSE environment. Syntactic errors are not covered, since they are trapped by the compiler. However since the eLSE environment was continually being enhanced, SIPDES could not be upgraded fast enough, therefore its use has been discontinued. There was also a big problem in knowledge acquisition to build up a database of the problems, since users tended to forget the problems/solutions they encountered after the event.

Section 2.2.3 : AI CONCEPTS IN SIMULATION

There are a number of AI concepts which have been used in simulation. Most notably, knowledge-based simulations and Natural Language Understanding and Processing. The next two subsections highlight these concepts. It is worth repeating that nearly all of these interests are at the research level [Paul, 1991].

KNOWLEDGE-BASED SIMULATIONS

A knowledge-based simulation can be defined [O'Keefe and Roach, 1987] as the use of a knowledge-based framework, with the system being simulated represented within a typical knowledge structure (for example a number of rules). The inference mechanism commonly used with the knowledge structure is extended by the addition of a time-flow mechanism. This clearly differs from tradition simulation programs where some or even all the details of the model is contained in the simulation program, which is compiled before running. In a knowledge-based simulation, the knowledge structure itself can be analysed, manipulated and altered either manually, or automatically by an inference engine. The concept that knowledge based simulations is essential for integrated simulation support environments will be examined in section 2.3 below.

Section 2.2.1 highlighted production rules as one type of explicit knowledge-base used by both expert system and simulation practitioners. A second type of knowledge-base used is object-

orientated languages, including SIMULA [Birtwistle et al, 1979], the first object-orientated language which was initially designed as a discrete event simulation language, ROSS [McArthur et al, 1986], based around an English-like, object-oriented language implemented in Lisp, SIMKIT [Harmon and King, 1985] and T-Prolog [Futo and Szeredi, 1982], where a simulation system is constructed using Prolog clauses. There are also experimental systems using object-orientated languages, including HIRES [Fishwick, 1985], SIMYON [Ruiz-Mier et al, 1985], BLOBS [Middleton, 1986] and PROSS [O'Keefe and Roach, 1987].

A third type of explicit knowledge-base which is derived from the AI/expert system area is semantic networks, a graphical AI knowledge representation technique. Semantic networks were originally designed as a way to represent the meaning of English words [Rich, 1983], but section 2.3.6 and chapter 3 will propose that semantic networks be used both as a simulation knowledge-base and as a core for an integrated simulation support environment.

NATURAL LANGUAGE UNDERSTANDING AND PROCESSING (NLUP)

Natural Language Understanding and Processing (NLUP) is the technique of understanding the spoken or written sentence. The first generation systems began in the 50s with the appearance of translation systems. These concentrated on syntax (grammar) but ignored semantics (meaning), which severely limited their use. The

second generation systems in the 60s operated on 'toy' domains and were impressive in keeping up conversations. The third generation systems in the 70s were designed to be used in real-life applications, but were limited in terms of syntax and semantics (domain). The two most advanced third generation systems [Doukidis, 1987] which were applied to the problem formulation stage of a simulation are SPIF [Doukidis, 1985] and NLPQ [Heidorn, 1972]. NLUP can also be used in querying the results of a simulation. One such system was designed by the Carnegie Group (1986).

NLPQ accepts a wide variety of sentences, but on a very limited domain. SPIF can only be used in defining the logic of the model [Doukidis, 1987], where it controls the conversation, as opposed to accepting a sentence in any area of the simulation. This has the advantage of forcing the analyst and user to be very concise and methodical. However there were a number of problems which made the system impractical. One of these problems is that the user tends to get very bored with describing each entity and activity in the model in turn to the analyst, who then relates the feed back of the system to the user (a problem partly due to the limited syntactic structure of the sentences). Another limitation derives from its interface with AUTOSIM [Paul and Chew, 1987] interactive program generator, since once SPIF is used and its conceptual model output is added to in later stages of the simulation, rerunning SPIF would result in the loss of this added data.

In most of these systems, the underlying methodology is the same two stage translation, a syntactic analysis stage, to check the word sequences and to build a syntactic tree or equivalent, followed by a semantic analysis stage to build the final knowledge-base. For example :

After Barking, the Dog then bites the postman.

The syntactic analysis stage tries to find a syntactic pattern which matches the sentence, with the possible aid of a dictionary, such as :

AFTER [activity], the [entity] THEN [activity] the [entity]

However, there could be more than one possible match, even after the interface has eliminated the matches which are not consistent with the existing knowledge-base. Here the system could either choose the most likely pattern or ask the user which pattern is the correct one. Since there is also the possibility that the existing knowledge-base is incorrect, and an automatically eliminated syntactic structure was actually correct, the ability of the user to examine the 'eliminated' structures would be useful.

The semantic analysis stage depends on the type of knowledge-base used, as well as the scope of the domain of the system. As a result of semantic analysis, new knowledge gained from the sentence can be

added to the knowledge-base.

An NLUP can be either hard-coded or data-driven. In a hard-coded NLUP, the syntactic and semantic structures are embedded in a compiled program which can not be changed without laboriously changing and recompiling the program. A data-driven NLUP accepts the definition of the syntactic and semantic structures as a separate input and applies them to the sentence being examined.

The hard-coded approach was used in this research project's first attempt at NLUP and a prototype program called SEF (Semantic Formulator) was developed. In the latest prototype, SASIM, a data-driven NLUP was developed. The data-driven approach adds flexibility to the system because the syntactic definition, held in a text file, can be examined and altered by a simple word-processor, without recompiling the system.

Section 2.3 : INTEGRATED SIMULATION SUPPORT ENVIRONMENTS

Simulation research has recently began to emphasis the development of integrated simulation support environments [Rozenblit et al, 1990] [Henriksen, 1983] [Hu et al, 1989] [Hu, 1989].

This section highlights current research in this area.

Section 2.3.1 : The goal

Section 1.4 mentioned that the goal of an integrated simulation support environment is to integrate the four stages of a simulation. Thus the integrated environment would provide automated support starting at the initial problem formulation stage right through to output analysis, with the ability to iterate back to an earlier stage.

In order to build an integrated simulation support environment, it is essential to pursue the goal of explicit knowledge representation (ie knowledge-based simulation). This is because during any stage of development of a simulation model if some description of the problem is changed (e.g. arrival distribution of customers or addition of an entity), all other processes of the simulation environment must have automatic access to the new information. This shift from program to the model view is highlighted by Nance (1983).

The explicit knowledge-base however need not be limited to AI concepts (for example object-orientated languages), but can be achieved using a 4th generation approach (relational databases), or other techniques, such as using activity cycle diagrams. Explicit knowledge representation has the very important advantage that reasoning can be performed directly on the knowledge-base. These will be described in the next sub-sections.

The difficulties in achieving this aim are highlighted by Balci and Nance (1987):

"The complete set of requirements for developing [a simulation model development environment (SMDE)] poses a significant challenge to SMDE designers and implementers. Nevertheless, we are confident that the challenge can be met by way of an evolutionary development of SMDE prototypes".

Despite the extent of the challenge of building an integrated simulation support environment, these difficulties are increasingly becoming surmountable, especially with the technical/cost improvements in general purpose computers with integrated multi-media capability.

Section 2.3.2 : Activity cycle diagram approach

As mentioned in section 1.4, activity cycle diagrams (ACDs) are one of the four key diagrammatic methods for representing the conceptual model of a problem. The other three diagrammatic methods are Augmented Petri Nets, Event Graphs and GPSS block diagrams. ACDs and Event Graphs have the least number of concepts. ACDs are also claimed to be the easiest diagrammatic method to explain to clients [Paul and Ceric, 1990]. Thus the ACD's graphical nature and its transparency enable both the simulation practitioner and the client to understand each other and the system being modelled.

The ACD has been notably used by CAPS [Clementson, 1982] and HOCUS [Szymankiewicz, 1984] among others. In the latter case, it is possible to run a data-driven simulation straight from the ACD, without going through a program generation stage.

However a disadvantage of ACDs is that complex conditions for cooperation between objects in an activity are difficult to represent graphically [Pidd, 1988] [Szymankiewicz et al, 1988]. This disadvantage is offset by the availability of automatic program generators which take the ACD as input and allow the complex conditions to be programmed into the generated code. A sign of the usefulness of the ACD [Paul and Ceric, 1990] is that nearly all known automatic program generators based on a diagrammatic conceptual modelling method use ACDs.

However, the generation of code is fundamentally problematic in that it implies a single directional progress, while simulation is intrinsically iterative. This can be illustrated by the following scenario :

A conceptual model has been built and a program generator has then been used to generate the code. The code has subsequently been altered, manually, to customise the application (say around 30% new code [Crookes, 1987]). At this stage, a correction or enhancement to the conceptual model has been requested. The choices are either to alter the conceptual model and regenerate

the code (thus losing the manual customisation), or to manually change the code (a long and laborious task).

This illustrates some of the problems which are bound to be encountered by not strictly separating the part of the knowledge-base contained in the conceptual model (say 70%) and rest of the knowledge-base contained in the customisation rules (production rules). The customisation rules become irrevocably intermeshed in a computer program. This reinforces the argument that integrated simulation environments should aim for explicit knowledge representation, where if the conceptual model and customised rules were kept 'separate', the conceptual model could be altered at a very late stage, without the need to re-enter the customisation rules. This approach would thus eliminate the requirement of a program generator stage (although the program generation of small parts of the conceptual model may still be advantageous to provide a base on which the customised code can be added).

Section 2.3.3 : Database approach

The approach of using a database as a central knowledge-base for an integrated simulation support environment has been used by Reese and Sheppard (1983), El Sheikh (1987) and Balci and Nance (1987). The structured textual nature of a database, however, can present difficulties in representing complex conditions for cooperation

between objects. It also presents difficulties in the visualisation of the problem, as well as performance difficulties due to the large processor overhead of traditional relational databases.

Reese and Shepard (1983) developed the Simulation Software Development Environment (SSDE). This is built around a series of databases and a language. El Sheikh (1987) produced INGRESSIM which ran on a VAX minicomputer. An automatic program generator was added to INGRESSIM by Mashour (1989). Balci and Nance (1987) developed a research prototype called Simulation Model Development Environment (SMDE), based on the conical methodology [Nance, 1981], which ran on a colour SUN 3/160 graphical windowed workstation. Notably both INGRESSIM and SMDE used the Ingress relational database package as their central knowledge-base. In both cases they relied on a program generator to generate the executable code, in order to allow for the complex interactions between objects. This, as explained in section 2.3.2 above, implies a single directional progress. Ideally an integrated simulation support environment should be able to read the information from the relational database, together with any customisation rules (again stored as data) and then run the simulation directly without generating any code.

Section 2.3.4 : Object-orientated language approach

As has been mentioned in section 2.2.3, object-orientated languages have been widely used to develop simulation systems. Due to their explicit knowledge representation, it is natural that they are used as the central knowledge-base for a simulation support environment. Some of these environments are :

The Rand Object-oriented Simulation System (ROSS) [McArthur et al, 1986], is probably the first and most developed AI based simulation tool [O'Keefe and Roach, 1987]. It is based around an English-like, object-oriented language implemented in Lisp. The objects can receive and send messages which can be intercepted by the simulation practitioner. The simulation can be interrupted and examined at any time. ROSS has been used to develop a graphical military simulation system called TWIRL [Klahr et al, 1986].

Knowledge-based Design and Simulation Environment (KBDSE) [Rozenblit et al, 1990] is an integrated simulation support environment implemented in Lisp. There are two basic components for the system, a front end for the model construction process and an object-oriented simulator supporting the evaluation of hierarchical, multi-component models. Model specification is performed using the discrete event system specification (DEVS) formalism [Kim and Zeigler, 1987]. However, as investigated by Domingo (1991), formal methods are not very easy to use in the simulation environment,

notably due to their large textual nature, even to describe a simple simulation.

Other systems include SIMYON [Ruiz-Mier et al, 1985], which is an object-orientated language prototype and MAGEST [Oren and Aytac, 1985], which is a knowledge-based modelling and simulation system.

Section 2.3.5 : Expert Systems (production rules) approach

It has been shown by Flitman and Hurrion (1987) that an expert system can hold the controlling logic (production rules) for a discrete event simulation model. This is an important step forward, since the conceptual model of the problem could be contained in a form (e.g. ACD) which is independent of the controlling rules. A possibility, which is not highlighted by Flitman and Hurrion, is that of localised control, where the ACD could control most of the model running, without requiring explicit production rules, with the expert system containing the part of the total logic which is required to be manually customised. This can be achieved by using production rules which are attached to individual objects (mainly activities) which regulate their starting conditions, their effects and the finishing instructions. This AI technique is called Procedural Attachment [Rich, 1983].

Earlier, a simulation production-system (using a PROLOG simulation engine) was also developed at the University of Warwick, but their strategy of separating the controlling logic from the simulation system, while sharing some data, conforms to the structure suggested by O'Keefe (1986b). Thus the user benefits both from the features of existing simulation environments and from the additional benefits of using an expert system to aid in achieving the desired "needs, goals and objectives" [Shannon, 1985].

Other simulation production-systems have been developed by Robertson (1986) and Goodman et al (1987). These are along the same lines as the Prolog based simulation system developed at the University of Warwick.

Section 2.3.6 : Semantic network approach

Semantic networks are a graphical AI knowledge representation technique where information is represented as a network of nodes (or objects) with relationships between them expressed by labelled and directed arcs (or Links). A detailed description of semantic networks is contained in the next chapter. A good description of semantic networks can also be found in Rich (1983) and Shirai and Tsujii (1984).

Semantic networks were first proposed by Quillian (1968) and Raphael (1968) for representing the meaning of English words in Natural Language Understanding and Processing (NLUP) systems. The only record of an attempt to use a semantic network approach to simulation is Knowledge-based Simulation System (KBS) [Baskaran and Reddy, 1984], renamed Simulation Craft. KBS enables interactive model creation and alteration, simulation monitoring and control and graphical display. It is based around a schema representation language [Reddy and Fox, 1982]. This is a knowledge-representation language based upon frames and written in Lisp. Frame representation structure is a technique which uses pre-designed semantic networks as a knowledge-base for a specific problem-solving task. The approach then applies a slot-and-filler algorithm to super-impose the pre-designed semantic network knowledge-bases (represented in multiple frames) onto the real world system to find the frame that fits best, thus 'understanding the system'. Typically a single frame describes a class of objects, e.g. DESK or ROOM. Multiple frames are linked together into a frame system to represent complex environments. 'Frame theory' was invented by Minsky (1975) at MIT and discussed in Kuipers (1975). However frame theory differs from the semantic network approach because it is specific to the class of model which the frames are built for, and therefore not as flexible as a general semantic network where the semantic network can expand with new structures.

However KBS's applicability to discrete event simulation is limited because it was firstly implemented as a textual database (as opposed to graphical), and secondly it provides a conceptual view of a simulation (similar to system dynamics) and not a discrete event system [O'Keefe and Roach, 1987]. KBS, instead, places considerable emphasis on introspection (where a simulation model learns about itself) and other methods of automatic analysis, without running a time-based simulation.

Section 2.4 : CONCLUSION

From the above analysis, it is possible to speculate what an ideal simulation environment should do and which aspects of both artificial intelligence and non-AI tools and techniques are probably appropriate to achieve these aims.

The main aim of our ideal simulation environment is to create an integrated simulation support environment which would provide automated support starting at the initial problem formulation stage right through to output analysis, with the ability to iterate back to an earlier stage. This environment should be built around a central explicit knowledge-base which should be able to be viewed and modified using a consistent graphical interface. The knowledge-base should be able to hold complex interactions between objects using procedural attachment of production rules (ideally passive

rather than active production rules). Using the consistent graphical interface, it should be possible to create, view and/or enhance the underlying knowledge-base using different representation techniques, including natural language, production rules and activity cycle diagrams. Since a simulation should be built around a single central knowledge-base, once the knowledge-base is updated using one representation technique, all views of the knowledge-base using other supported representation techniques should be updated automatically. Some representation techniques, for example natural language, may however be most useful as an input mechanism, rather than an output mechanism. If desired by the user, it should be possible to restrict a view to sub-parts of the knowledge-base, thus possibly avoiding the display of information not immediately relevant to the task the user wishes to perform at that time. It should be possible to automatically check the knowledge-base for internal anomalies and inconsistencies (to aide validation and verification of the simulation model). There should also be an inference engine which should be able to trace the links between objects in the knowledge-base (e.g. spot direct and indirect interactions between entities). The inference engine should be able to take into account the production rules held in the knowledge-base using procedural attachment. It should be possible to perform an actual simulation run, even if only part of the knowledge-base is entered, thus providing some feed-back at an early stage. During the simulation run, it should be possible to view the entities

moving around the system using any of the supported representation techniques (e.g. activity cycle diagram view). This would thus make it possible to have a consistent view of the simulation while both creating and running the simulation model. It should be possible during a simulation run to view the histograms of any of the queuing times and queue length, and if desired, to change the knowledge-base during a simulation run. It should be possible to specify custom histograms and also to automatically analyse captured data from the simulation run.

Ruiz-Mier et al (1985) speculate on the future use of a network simulation language in an AI programming environment. This is the area which this research project tackles. Since an ACD can be regarded as a network of nodes (or objects) with relationships between them expressed by labelled and directed arcs (or Links) [a definition of semantic networks], the semantic network knowledge-base seems a natural area for further investigation. The fact that semantic networks were first used in natural language processing is an added advantage, since this is one of the areas of research in the problem formulation stage of a simulation. These ideas led to the development during this research of semantic modelling, a discrete event simulation modelling approach based on semantic networks. Semantic modelling is described next in chapter 3.

CHAPTER 3

SEMANTIC MODELLING : DESIGN ISSUES

Section 3.1 : INTRODUCTION

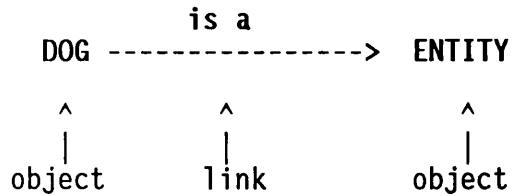
The previous chapter indicated that the semantic network knowledge-base requires further investigation. This chapter explores this knowledge-base to see if it can be adapted to the simulation environment. Section 3.10 demonstrates semantic modelling, a discrete event simulation modelling approach based on semantic networks, which attempts to achieve the aims, highlighted in the previous chapter, of what an ideal simulation environment should do.

Chapter 4 describes a prototype implementation of semantic modelling.

Section 3.2 : UNDERLYING STRUCTURE OF SEMANTIC NETWORKS

Semantic networks are a graphical AI knowledge representation technique. Information is represented as a network of nodes (or objects) with relationships between them expressed by labelled and directed arcs (or Links). The traditional application of semantic networks in the AI community is for Natural Language Understanding and Processing (NLUP).

An example of a very small semantic network is



("is a" is a relationship and it can be involved in many links)

Using this structured representation, networks can be built up expressing relationships between objects, together with their properties. The networks can be applied to many fields, each requiring specific customisation, but this project concentrates on the discrete event simulation field. However semantic networks provide no automatic internal structure, since an individual node can contain any type of data (possible a single word, or a paragraph, or a whole encyclopedia, or a visual image). In addition each relationship, regardless of what it is called by the person who entered it, can mean different things to different users, and may not be understood at all by a computer program trying to understand the semantic network knowledge-base. Thus I have had to impose a structure on the semantic network to convey meaning applicable to a discrete event simulation. These will be highlighted in the next subsections.

Section 3.2.1 : Synonyms

Synonyms occur when different words mean the same thing, for example "Nine" and "9", "Drinking" and "Drink", "Leave" and "Exit", "For" and "Duration", "Served" and "Service", "Customers" and "Customer". This would clearly cause problems, for example when trying to decide whether an object being described by a client is a new object or a further description of an existing object. I propose that all plurals should be automatically converted to the singular (e.g. "Customers" converted to "Customer") and all numbers to be stored as the arithmetic number (e.g. "Nine" converted to "9") and all verbs converted to the 'base' verb (e.g. "Served" converted to "Service"). It should be noted that where different 'base' verbs are possible, e.g. "Service" or "Serve" or even "wait on", the actual base verb stored in the semantic network is not important, as long as it is used consistently throughout the semantic network (i.e. there should not be two nodes where one is called "Service" and the other "Serve", where they both refer to the same activity). This conversion of synonyms would need to be done automatically. Since there may be different programmes accessing and updating the knowledge-base, they should use the same synonym conversion algorithm to avoid inconsistencies.

Section 3.2.2 : Core objects, instances and the "IS A" relationship

I have defined three core objects, these are "ACTIVITY", "ENTITY" and "DECISION". An object on the semantic network can then be linked to one of these core objects using the "IS A" relationship.

e.g. "POSTMAN" IS A "ENTITY"
 or "BITE" IS A "ACTIVITY"
 or "IF IT IS RAINING" IS A "DECISION"

An object can only have one "IS A" relationship, ie an object can not be both an activity and an entity. If an object does not have an "IS A" relationship, it is still held on the semantic network, but assumed to be of an undefined type. Once these relationships are defined, a query such as "list all known activities" can be done by finding all objects which have a "IS A" "ACTIVITY" link. The decision core object is described further in section 3.2.3.

A further important concept is that of an instance of an entity. This is an extra defined object for every activity which an entity gets involved in. For example, if a customer might drink and he might rest, there would in addition to the "CUSTOMER" IS A "ENTITY" link, be two extra instances "CUSTOMER (DRINK)" and "CUSTOMER (REST)". These two objects are instances of "CUSTOMER". These 'extra' instances are required in order to be able to express individual facts about entities involved in activities, such that as more than one entity is needed in an activity or to express the cycle of an entity. The instances would be linked to the master

entity with the "IS A" relationship as follows :

"CUSTOMER (DRINK)" IS A "CUSTOMER"
and "CUSTOMER (REST)" IS A "CUSTOMER".

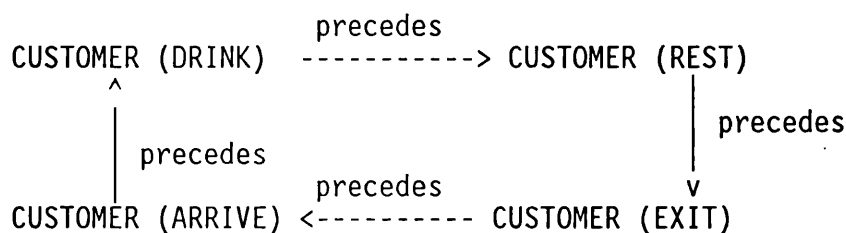
The actual name of the instances, for example using "SLEEPER" instead of "CUSTOMER (REST)" is not important, but by default, I use the entity name with the activity name in brackets.

Section 3.2.3 : The "PRECEDES" relationship

The "PRECEDES" relationship defines the cycle of an entity. For example, if a customer drinks and then he rests, the "PRECEDES" relationship would indicate this by the following link :

"CUSTOMER (DRINK)" PRECEDES "CUSTOMER (REST)"

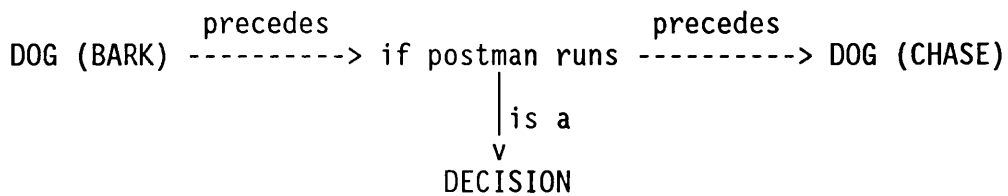
In an entity cycle, the "PRECEDES" relationship would create the cycle. For example :



To conform to the standards of an activity cycle diagram, all cycles of entities must be closed. Therefore the "CUSTOMER (EXIT)" PRECEDES "CUSTOMER (ARRIVE)" was added to close the Customer's

cycle. It is worth noting that, as opposed to an activity cycle diagram, there are no explicit queues, but in my semantic network approach each instance has an implicit queue associated with it.

For a decision (ie for a condition), the "PRECEDES" relationship would indicate the flow of the instance, e.g. 'IF THE POSTMAN RUNS, THE DOG WILL CHASE' would be expressed as :



Section 3.2.4 : The "DURATION" relationship

The duration relationship defines the length of time an activity takes. For example,

"REST" DURATION "3"

"DRINK" DURATION "10"

"ARRIVE" DURATION "NEGEXP(10,5)"

For the simulation being modelled, the time unit (seconds, minutes, hours etc.) would be consistent for all links which use the duration relationship. There is no constraint on the distribution used, such as negative exponential, normal or poisson, as long as the simulation running process can understand it.

Section 3.2.5 : The "NUMBER" and "INIT NUMBER" relationship

A discrete event simulation requires that the number of each entity there is in the system is defined (with the possibility that some entities are indefinitely large). This information would be contained using the "NUMBER" relationship which would be referred to from the master entity. For example, if there are ten glasses in the system, this would be expressed as :

"GLASS" NUMBER "10"

Additionally, at the start of a simulation the individual entities may be distributed throughout the system, as determined by the user. This information would be contained using the "INIT NUMBER" relationship which would be referred to from the instances of the entity. For example, if, at the start of simulation, seven of the above ten glasses are ready to use in the service activity and the other three are waiting to be washed, this would be expressed as :

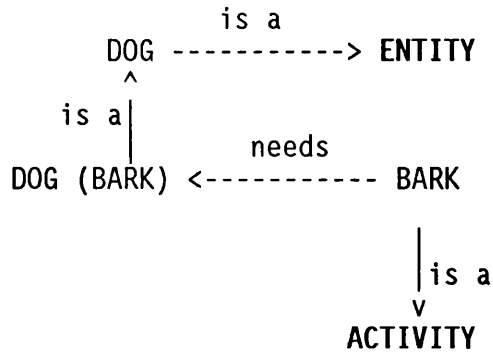
"GLASS (SERVICE)" INIT NUMBER "7"

"GLASS (WASH)" INIT NUMBER "3"

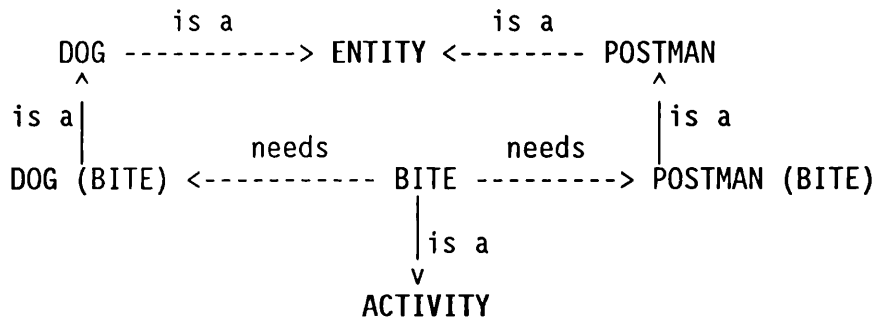
As part of the validation check, the total of the "INIT NUMBER" relationships for an entity should be equal to the total as defined by the "NUMBER" relationship. Certain entities which operate as central facilities, for example barmaids who either wash and serve depending on demand, would only require a "NUMBER" relationship.

Section 3.2.6 : Examples

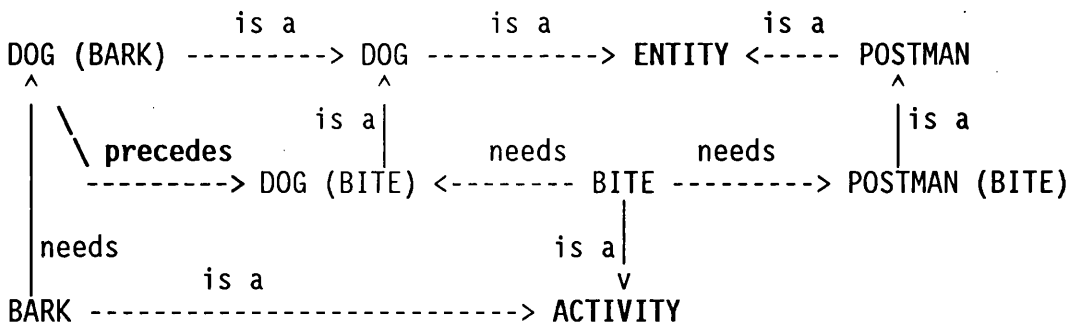
'THE DOG BARKS' would be expressed as :



'THE DOG BITES THE POSTMAN' would be expressed as :



'AFTER BARKING, THE DOG THEN BITES THE POSTMAN' would be expressed as :



This new semantic network is created by the 'merging' of the two

simpler semantic networks shown earlier, with the addition of the following link between the instances DOG (BARK) and DOG (BITE) :

DOG (BARK) ^{precedes} -----> DOG (BITE)

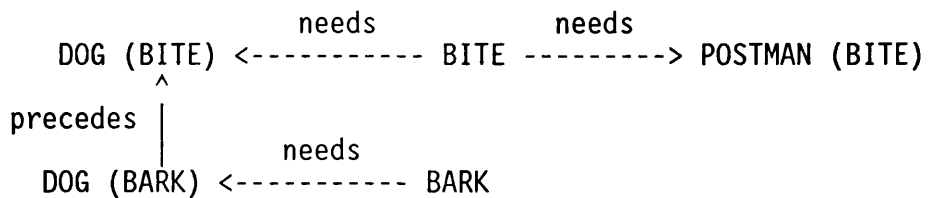
Section 3.3 : GRAPHICAL INTERFACE

As can be seen from the previous section, a semantic network for even a relatively simple simulation can not be easily represented in two dimensions, whether on paper or on a two dimensional computer screen, since the links would overlap too often, impairing understanding of the network. There are two complementary approaches to getting round this problem, where the actual network is stored internally as a single knowledge-base but is represented on the screen as multiple smaller knowledge-bases.

The first approach is to use vertical decomposition whereby there is a global picture whose elements could be examined in detail. At this more detailed (lower) level, there are also elements which can themselves be examined in further detail, and so-on. This could be visualised as breaking down a single large semantic network knowledge-base into smaller semantic networks (for example a hospital can be split between in-patients and out-patients).

The second approach relies on the distinctive classes of the relationships found in a semantic network. If only selected

relationships (and their associated objects) are displayed, it may be easier to ascertain information relating to those specific relationships (at a cost of not seeing the other relationships). For example, if the "IS A" relationship is hidden from the semantic network translation of the statement 'AFTER BARKING, THE DOG THEN BITES THE POSTMAN' (shown in section 3.2.6), the sentence would be expressed graphically as:



This is much more compact and readable (ie more transparent) than the semantic network shown in the previous section. (Note that we do not need to display the objects "Entity", "Activity", "POSTMAN", "DOG" since they are not linked to any other objects on this graph). One way of visualising the splitting up of the semantic networks into distinctive classes of relationships is to think of each class as a different "level" of one large network, with the ability of moving between different levels, by means of a "lift". A link, relationship or object can be on more than one level. Where an object exists on more than one level, the lift mechanism can then be used to move the user from one level containing a specific object directly to all other levels containing that object, while skipping levels not containing it. This could be helpful when examining all knowledge held about an object.

Having 'split' up the large semantic network into smaller units, which because of their smaller size should be more manageable than the original single network, it is desirable to use the power of modern computer graphics to display the objects as "icons" on a screen with the links being displayed as arrows. The name of the links (ie the relationship name) can be displayed along the arrow (similar to the semantic networks described earlier). Links can be added (or deleted) by simply pointing to the two objects (for example using a mouse) and typing in the relationship name (or highlighting the relationship name from a menu of all known relationships). The ability to move quickly around the semantic network is essential, together with the ability to modify any object or relationship name by simply highlighting the name on the screen (for example using a mouse) and editing it. The ability to ZOOM OUT or ZOOM IN, in order to see a fuller or more detailed view of a level, is also desirable.

The graphical level also needs to contain the main menus of the system since the user should always start from the graphical interface when selecting any of the other facilities of the system. This is in keeping with the idea that the semantic network can be used as the central knowledge-base in an integrated simulation support environment.

Section 3.4 : NATURAL LANGUAGE UNDERSTANDING AND PROCESSING

The previous section highlighted that, by splitting the semantic network, it may be possible to make the network more transparent to the user. However this brings up the problem of maintainability. How do you add new links when the objects may be on different levels? A further problem, as seen above, is that a simple piece of knowledge may require multiple links in order to be expressed in a semantic network. What is required are concepts which take simple inputs (in whatever form) and automatically translate them into the semantic network. There is potentially a large number of such concepts, but I will examine two concepts to try and overcome these problems. Firstly an input facility for activities and/or entity cycles (for example working from an Activity Cycle Diagram) which is highlighted in section 3.5. Secondly a Natural Language Interface which can take a 'standard' english sentence and translate it into the semantic network format.

In order to translate an English sentence, there are a number of hurdles to overcome. They can be expressed as follows :

- (a) Superfluous words in a sentence (for example "A", "Also", "At").
- (b) Different words meaning the same thing, for example "Nine" and "9" or "Drinking" and "Drink" or "Leave" and "Exit" or "For" and "Duration" or "Served" and "Service".

- (c) The phrases in a language have to be expressed to a computer.
- (d) Sentences can have different meanings depending on the context and the environment.

Problems (a), (b) and (c) can be regarded as **syntactic**, while problem (d) can be regarded as **semantic**.

Most Natural Language Understanding and Processing (NLUP) systems take a two stage approach : Firstly, syntactic analysis to build a number of syntactic structures (trees) representing the sentence. Secondly, semantic analysis which takes each possible syntactic structure and tries to eliminate the ones which are not consistent with the knowledge-base.

Section 3.4.1 : Syntactic Analysis during NLUP

The first stage of syntactic analysis is to split the sentence into separate words and to translate them to some 'core' set of words, using a dictionary. The core set of words is dependent on the domain. For example :

Hundred	->	100
Also	->	<ignore>
A	->	<ignore>
For	->	Duration

Drinking -> Drink

Barmaids -> Barmaid

There will tend to be rules for some of these, for example translating plural into singular (for example "*MEN" to "*MAN" or "*S" to "*") or past tense verbs to present tense (for example "*ED" to "*" or "*ING" to "*"). However to every rule there are exceptions (for example "*ERVING" to "*ERVICE" or "AS" to "AS"), and so the system should be able to be told the rules together with the exceptions in order to minimise the size of the dictionary.

The second stage of syntactic analysis is to build all possible syntactic structures which the sentence conforms to. This raises two problems, the first being the expression of the syntactic structures (phrases) in the language, the second being how these syntactic structures are mapped onto a semantic network. This second stage of syntactic analysis should be performed as follows: A sentence would be defined as a number of possible structures, for example

KEY ^ indicates start of a keyword
* indicates beginning of syntactic structure (phrase)
; indicates ending of syntactic structure (phrase)
~ indicates optional word or structure
/ indicates alternate structure
> semantic translation of phrase
" " indicates a group of words to be treated as a label

Syntactic Structure

Notes

*SENTENCE :

Definition of "Sentence"

^THERE ^IS ENTITY_P "^. "

One possible structure

>

(always end with a \.')

/

or

^THERE ^IS ACTIVITY_P "^. "

Another structure

>

;

End of structures

where ENTITY_P and ACTIVITY_P are substructures defined as :

*ENTITY_P :

Definition of "ENTITY_P"

~obj_number obj_name

"obj_number" optional

> obj_name ^NUMBER obj_number,

e.g. BARMAID NUMBER 10

obj_name "^IS A" ^ENTITY

e.g. BARMAID IS A ENTITY

/

or

~obj_number obj_name ~^WHICH ~^IS ~^AN ^ENTITY

Another structure

> obj_name ^NUMBER obj_number,

e.g. BARMAID NUMBER 10

obj_name "^IS A" ^ENTITY

e.g. BARMAID IS A ENTITY

;

End def. of ENTITY_P

*ACTIVITY_P :

Definition of ACTIVITY_P

obj_activity

One possible structure

> obj_activity "^IS A" ^ACTIVITY

e.g. WASH IS A ACTIVITY

/

or

obj_activity ~^WHICH ~^IS ~^AN ^ACTIVITY

Another structure

> obj_activity "^IS A" ^ACTIVITY

e.g. WASH IS A ACTIVITY

;

End def. of ACTIVITY_P

The sentence "There is one drink." would thus only have one syntactic structure based upon the above structures. The mapping of this onto the semantic network would be :

"DRINK" NUMBER "1"
and "DRINK" IS A "ENTITY"

However, the sentence "There is a drink." would have two syntactic structures. The mapping onto the semantic network being :

(a) DRINK IS A ENTITY (ie based on first sentence structure)
or (b) DRINK IS A ACTIVITY (ie based on second sentence structure).

Semantic analysis, described in the next section, is required to clarify which is the correct translation.

Section 3.4.2 : Semantic Analysis during NLUP

Syntactic analysis results in a number of possible syntactic structures. It is the role of semantic analysis to reduce these to a single syntactic structure, which is then used to update the knowledge-base.

This can be achieved by following four steps. Firstly, the elimination of all structures which are not consistent with the existing knowledge-base. For example if it is known that "DRINK IS

A ACTIVITY", any structure which stated that "DRINK IS A ENTITY" can be eliminated. Secondly, the elimination of all structures which are not internally consistent (i.e. as previous step, but comparing the separate phrases in the sentence). Thirdly, using probability theory where appropriate to rank the possibilities (this step was not implemented in the prototype implementation because it would require the syntactical structure definition to indicate the frequency of use of each syntactic structure in a typical sentence). The fourth and final step, where there is still uncertainty, would be to ask the user which of the translations is correct. These four steps should result in a single syntactic structure, which is then used to update the knowledge-base.

The above steps assume that the existing knowledge-base is correct. A possible enhancement to the semantic analysis stage would be to provide a mechanism to override the existing knowledge-base with new 'contradictory' information, and delete the parts of the existing knowledge-base which are not consistent with the new information.

Section 3.5 : ACTIVITY AND ENTITY CYCLE INPUT

As highlighted in section 3.4, the activity and entity cycle input concept takes a graphical input of activities and/or entity cycles (useful, for example, when the problem has been already described using as Activity Cycle Diagram) and automatically translates it into either a new semantic network, or adds it to an existing semantic network.

Two approaches need to be handled, either tracing each entity around its cycle or choosing an activity and identifying the entities involved in it. This would enable the easy introduction of new entities, instances, activities and links.

The entry of activities or entities would be aided by using a combination of a mouse to select the objects from a screen and using a keyboard to enter the name of new objects which are not presently represented on the network. However mouse support was not implemented in the prototype implementation because of the lack of a mouse support capability in the programming environment used.

Section 3.6 : PROCEDURAL ATTACHMENT OF PRODUCTION RULES

As noted in section 2.3.2, complex conditions for cooperation between objects in an activity are difficult to represent graphically. Active production rules, as highlighted in section

2.2.1, have been used in simulation to hold the conditions for cooperation between objects. It is therefore a requirement to utilise production rules on a semantic network but without detracting from the semantic network being the central knowledge-base. As section 2.3.5 highlighted, the procedural attachment technique enables production rules to be attached to individual objects (mainly activities) which regulate their starting conditions, their effects and the finishing instructions.

Production rules can be implemented as either passive (i.e. totally data-driven) or active (i.e. contained in program code which is compiled). Passive production rules would be enterable from a keyboard by first moving to the object to which the rules are to be attached, and then typing them in. It would then be possible to examine any of these production rules by selecting any object on the semantic network. Its production rules, if any, would then be displayed in a window on the screen and they can then be changed by editing them. In addition, since they are passive, they can then be used by the inference engine described in section 3.8. It is therefore clearly desirable to use passive production rules, as opposed to active production rules, although they are much harder to implement. The prototype implementation uses active production rules because it aims to illustrate the concept of procedural attachment, and the required extra effort to implement passive production rules would not further the aims of this thesis.

Two possible problems arise from the procedural attachment approach. Firstly it is not a graphical knowledge-base and therefore may be harder to integrate into the graphical interface showing semantic networks, especially since concepts which manipulate and analyse the semantic network knowledge-base would not automatically be able to do the same with production rules. Secondly, in some problems, for example the port problem, described by El Sheikh et al (1987), the activity cycle diagram conceptual model (and therefore the semantic network implementation) is very simple, but these are problems because of their very complex rules and conditions (e.g. rules for loading and unloading). These, in a semantic network implementation, would be implemented using procedural attachment, which in these complex examples would relegate the importance of the actual semantic network since the procedural attachment would control most, if not all, the simulation run.

Section 3.7 : VALIDATION AND VERIFICATION

Validation is the process of checking that the conceptual model corresponds to the real life process. Since, in the semantic modelling approach, the conceptual model (knowledge-base) is held either directly using a semantic network or indirectly using procedural attachment, it is desirable to validate it. Since the knowledge-base is also the computer model, the verification that the

computer model corresponds to the conceptual model is simplified, since in traditional simulation systems most errors from going from the conceptual model, contained on paper, to the computer model are caused by incorrect manual input of the conceptual model into the program generator.

The validation process would detect a number of anomalies (not necessarily errors), including :

- (a) Entities which do not complete a cycle.
- (b) Entities which are superfluous (do not take part in any activities).
- (c) Semantic networks which are not cohesive. This would occur if it is possible to separate the network into a number of independent sub-networks. This would imply that the domain is two separate simulation domains and not one.
- (d) Lack of information on entities (for example number of entities, starting locations in a simulation etc).

If appropriate, this stage can actually enhance the knowledge-base with additional links (for example in order to complete a cycle).

Section 3.8 : INFERENCE ENGINE

Having constructed, validated and verified a knowledge-base, there are a number of approaches to infer extra knowledge about the domain. Two of the possible approaches are either a quantitative

approach, ie running a traditional simulation (described in section 3.9), or a qualitative approach using an inference engine. This section will highlight such an inference engine for the qualitative approach which is derived from using expert system techniques.

The graphical nature of semantic networks lends itself to logical inferencing, where the links between two objects can be qualitatively ascertained by finding the 'chain of relationships' between two objects in the network, using searching algorithms, then attempting to draw some conclusions based on the knowledge of the types of objects and relationships along the chain. This latter stage depends on some 'in built' knowledge of what some relationships and objects mean, such as the relationship 'PRECEDES' is a one-way relationship, while 'NEEDS' is a two-way relationship. It is important to stress that this qualitative information is used to complement the quantitative information resulting from an experimental simulation run using the same semantic network.

The three main search algorithms are the same as the shortest path algorithms used to find the shortest path through a network of nodes, without visiting the same node twice. These are :

1) DEPTH FIRST

Form a NODESSTACK consisting of only one NODE, being the starting NODE.

REPEAT

If the NODESSTACK is empty, announce destination can not be reached, and STOP.

Else If the first NODE in the NODESSTACK reaches the destination SIGHT, announce destination reached, with the route and distance, and STOP.

Else remove the first NODE from the NODESSTACK and add all the NODES, which are children of the discarded NODE, if any, to the front of the NODESSTACK. If progress required, display discarded NODE.

2) BREADTH FIRST

Form a NODESSTACK consisting of only one NODE, being the starting NODE.

REPEAT

If the NODESSTACK is empty, announce destination can not be reached, and STOP.

Else If the first NODE in the NODESSTACK reaches the destination SIGHT, announce destination reached, with the route and distance, and STOP.

Else remove the first NODE from the NODESSTACK and add all the NODES, which are children of the discarded NODE, if any, to the

back of the NODESSTACK. If progress required, display discarded NODE.

3) BRANCH AND BOUND

Form a NODESSTACK consisting of only one NODE, being the starting NODE.

REPEAT

If the NODESSTACK is empty, announce destination can not be reached, and STOP.

Else If the first NODE in the NODESSTACK reaches the destination SIGHT, announce destination reached, with the route(s) and distance, and STOP.

Else remove the first NODE from the NODESSTACK and add all the NODES, which are children of the discarded NODE, if any, to the NODESSTACK, position being consistent with maintaining a NODESSTACK with the least distant nodes at the front, (while if one of the NODES reaches the destination, this is added as far down the NODESSTACK as possible, while still maintaining a sorted list.

Branch and Bound is the slowest, due to the requirements to maintain a sorted stack but it is the only algorithm that guarantees optimality. Depth first can sometimes find a quick route between objects, but it can be diverted into 'dead ends'. Breadth First systematically searches all routes, always hitting on the route with

least number of links, without regard to any weights (or distances) on the route.

In semantic networks, most distances are the same, except when logic dictates that a relationship is one way, or that the relationship is irrelevant, whence the effective weighting of that link, in the direction being investigated, is set to infinity. Since links can be weighted, and the requirement is always to propose the nearest causal relationship first, Branch and Bound is the best algorithm to use in the inference engine. This is the approach used in the prototype implementation, with all links being the same weight (bar one way links).

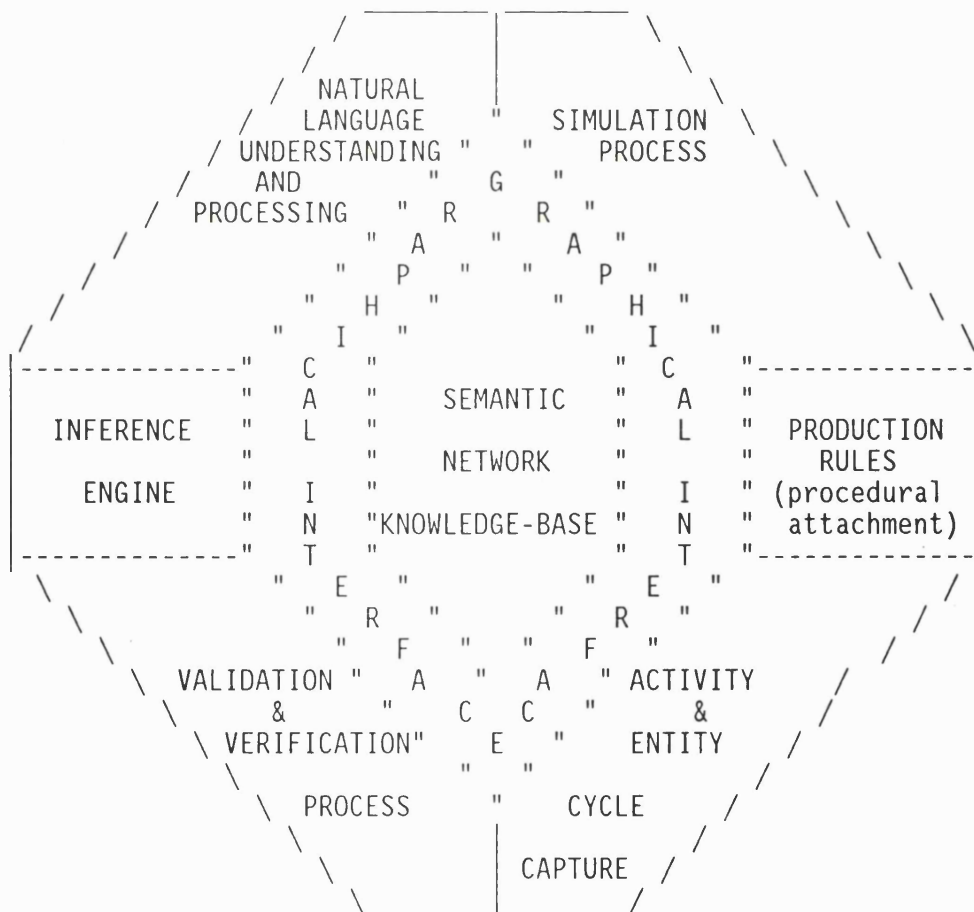
Section 3.9 : SIMULATION RUNNING

Since semantic networks are a graphical knowledge-base, it lends itself ideally to a graphical interface during a simulation run. During a simulation, objects move from one activity to a waiting queue before starting another activity. This motion can be indicated with an 'icon' moving from one instance of an object to another instance, for example CUSTOMER (DRINK) to CUSTOMER (REST). It would also be desirable for some activities to be indicated as a conveyor belt with icons moving along the belt. The number of entities waiting in a queue for an activity can be indicated by a number displayed above the instance object. The number of entities actually involved in an activity can be indicated by a number displayed below the instance object. Histograms of waiting times, queue length and time series can be viewed by selecting any object from the semantic network, even DURING a simulation run. Results can be viewed on the screen or output to a file. With this information, it may be possible to spot bottlenecks as well as to use the results as a basis for further study.

In theory, the semantic network knowledge-base can be altered even during a simulation run, though this may present some implementation problems.

Section 3.10 : SEMANTIC MODELLING ARCHITECTURE

The previous sections described semantic networks, together with various concepts which try to make the network manageable and maintainable. This section describes how these concepts can be assembled together to create a discrete event simulation modelling approach, called semantic modelling, which uses semantic networks as the central knowledge-base to an integrated simulation support environment which attempts to give a consistent graphical interface throughout the life cycle of a simulation study. The semantic modelling approach is best described by the following diagram :



"" = INTERFACE BETWEEN SECTIONS

As can be seen from the above structure, the semantic network is used as the central knowledge-base, with the graphical interface acting as the user interface between the various concepts and the central knowledge-base.

The semantic modelling approach is an extension of the approach used by the Simulation Model Development Environment (SMDE) prototype developed by Balci and Nance (1987), which is based on the conical methodology [Nance, 1981]. A key difference between semantic modelling and SMDE is that SMDE uses the INGRES relational database as the central knowledge-base, instead of semantic networks. SMDE similarly has a number of tools built around the knowledge-base, including a Model Generator, Model Analyser (using AI techniques) to validate the model and Model Verifier which analyses the executable program.

Section 3.11 : CONCLUSION

This chapter has described both semantic networks and semantic modelling, a discrete event simulation modelling approach based on semantic networks, which attempts to give a consistent graphical interface throughout the life cycle of a simulation study. The five main AI techniques used in semantic modelling are the semantic network graphical knowledge-base, data-driven natural language understanding and processing, procedural attachment, expert systems

and shortest path algorithms. Since the approach is an open architecture, more techniques, possibly from other research areas, could be added in future to improve the simulation environment.

Chapter 2 speculated what an ideal simulation environment should do. The main aim was to create an integrated simulation support environment which would provide automated support starting at the initial problem formulation stage right through to output analysis, with the ability to iterate back to an earlier stage. Semantic modelling attempts to do this by using semantic networks as the central explicit knowledge-base which could be viewed and modified using a consistent graphical interface. Complex interactions between objects could be held using procedural attachment of production rules. Using the consistent graphical interface, it is possible to create, view and/or enhance the underlying knowledge-base using different representation techniques, including natural language, production rules and activity cycle diagrams. Another of the highlighted aims was that once the knowledge-base is updated using one representation technique, all views of the knowledge-base using other supported representation techniques should be updated automatically. In semantic modelling, this is achieved by using semantic networks as the central knowledge-base, for example knowledge entered as natural language could be viewed and enhanced using the activity cycle diagram representation. The aim to restrict the view of users to sub-parts of the knowledge-base, was

achieved by using the concept of levels. The aim of checking the knowledge-base for internal anomalies and inconsistencies was achieved by the Validation and Verification process. The requirement of an inference engine is also included in semantic modelling, but since the production rules implemented in the prototype were active, the inference engine could not take them into account. The requirement to perform an actual simulation run, with the ability to view the entities moving around the system, together with their histograms, using the same views as seen when creating the knowledge-base, was also included in semantic modelling.

However one of the aims has not been supported. There is no support for the automatic analysis of captured data from the simulation run.

It is possible to note some potential problems with the semantic modelling approach. Firstly, there may be a problem with the size of the network. As shown in section 3.2, even a simple simulation can generate many semantic network links. These would need to be displayed 'intelligently'. This means enabling the user to view just the links he is investigating, and not to overload the screen with intercrossing links. This size problem leads to a number of related problems, which are described below.

Secondly, there may be a problem with the physical screen size. Following from the size of the network, it is obvious that a large

high resolution screen (and supporting software platform) would be very advantageous. The IBM-PC family initially had the Colour Graphics Adapter (CGA). This was superseded first by the Enhanced Graphics Adapter (EGA) then by the Video Graphics Array (VGA). The VGA itself has been superseded by a number of very high resolution displays, including Extended Graphics Array (XGA), and other non-standard adapters (over 1024x768 pixels - 256 colours). This again indicates that technical improvements are bound to make semantic networks more implementable (but not necessarily more understandable by the user).

Thirdly, there may be a problem with the internal complexity of the implementing software. This is an unavoidable problem since the aim of the approach is to create an integrated simulation support software. However the choice of a standard high level language and a standard hardware architecture will ease the problem of building and supporting the system, especially as different programmers will be responsible for support and development over time.

Fourthly, there may be a problem with the ease of use of the software. The complexity of the software (and its radical approach) could present a conceptual challenge to the user. This can be best overcome by adopting standard interfaces (e.g IBM's SAA interface, Windows/Macintosh mouse interface or a Lotus 1-2-3 like menu interface).

Despite these potential problems, semantic modelling may make a contribution to the discrete event simulation community. The next chapter describes a prototype implementation of semantic modelling. Chapter 5 shows how procedural attachment is implemented in this prototype. Chapter 6 critically analyses both the semantic modelling approach and the prototype.

CHAPTER 4

SASIM PROTOTYPE : IMPLEMENTATION ISSUES

Section 4.1 : INTRODUCTION

The previous chapter has presented the semantic modelling approach. This chapter presents the workings of a prototype implementation called SASIM. This detailed description helps to advance the readers understanding of the practical implications of implementing the approach.

The implementation of procedural attachment in the prototype is described in chapter 5. Working trials of the prototype, together with a critical analysis of semantic modelling concept, are contained in chapter 6.

Section 4.2 : GRAPHICAL INTERFACE

SASIM runs on IBM PC's and compatibles and is designed around the familiar spreadsheet concept that is used in Lotus 1-2-3 and Excel. This interface was chosen to reduce the potential learning curve of the user of the system, since many users have used and understand the concept of spreadsheets.

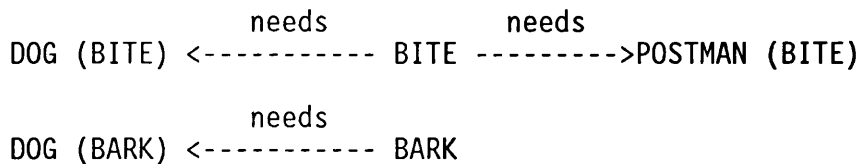
The requirement is to display and manipulate complex semantic models. These, as seen in the previous chapter, can resemble a road/rail network linking different nodes. If all links were shown at once then it would be very difficult to see and understand the model. Therefore, just like there are overview maps for main motorways, another for main railway links, and detailed maps for towns or counties, semantic networks can be split into multi-levels for ease of understanding. SASIM has a five level model as described below. There is no particular reason for there being only five levels, more levels can be added as appropriate. The five levels in the prototype represent a convenient division of the hierarchy into different logical or visual levels.

Each relationship is automatically allocated to one (and only one) level, and only objects which have a relationship at the current level are displayed on the screen. In a general semantic modelling implementation, a relationship could be on more than one level, but this would present additional implementation difficulties. The user

can subsequently also make an object appear on another level, but only the relationships of the current level will be displayed. This considerably reduces the number of links and objects displayed at any one time, and by logically bunching the links which are similar, it considerably increases understandability. The five levels used on SASIM are illustrated using the example sentence :

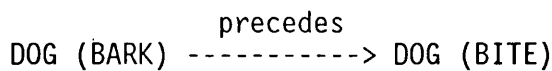
AFTER BARKING, THE DOG THEN BITES THE POSTMAN

0) BASE LEVEL (all relationships not falling under the other levels)



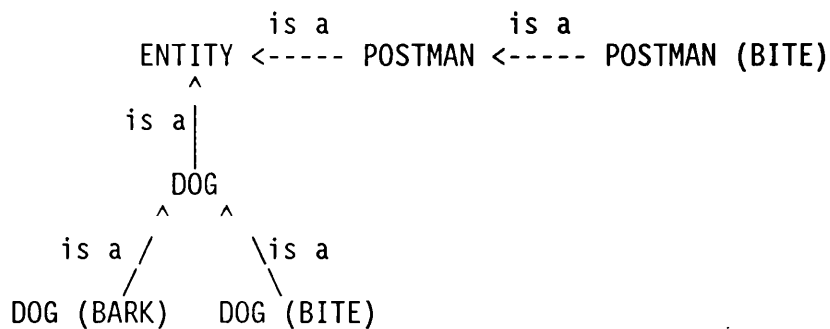
This is the level where each activity and the entities involved in that activity are shown.

1) ENTITY CYCLE LEVEL (Relationship PRECEDES)



This level shows the cycle of an entity, including conditional branching (decisions).

2) 'IS A' LEVEL (Relationship 'IS A')



This shows all decisions, activities, entities (and their instances).

3) NUMBERS LEVEL (Relationship NUMBER and DURATION)

This level would contain knowledge of how many dogs and postmen where in the simulation system :

for example number
 DOG -----> 10

This level would also contain the knowledge if 2 dogs where necessary to bite the postman (since he can cope with one dog without being bitten) :

 number
 DOG (BITE) -----> 2

4) USER LEVEL

This level contains objects, without any relational arrows. This enables flexibility in the graphical presentation. Objects must be located on this level using the graphical interface. This is essentially a notepad where selected objects can be placed for monitoring and/or manipulation.

Each object is automatically allocated a cell on at least one level, with relationships between objects shown by labelled and directed arrows. The interface enables the user to move the objects around to provide better transparency of the problem. In addition, when object names are edited on one level, all other occurrences of this object on other levels are automatically changed.

The first screen on starting the prototype is a blank spreadsheet. At this point, the user could choose between four ways to create a

semantic network knowledge-base (spreadsheet). Their first option is to retrieve a saved knowledge-base (spreadsheet file) using the multi-level menu system described in section 4.3 (accessed by pressing the '/' key). A second option is for the user to enter the activity cycle of an entity, as described in section 4.5 (accessed by pressing the [F6] key). A third option is to enter an English sentence for translation, as described in section 4.4 (accessed by pressing the [F8] key). A fourth option is to type in each object name in different cells on the spreadsheet and then use the [F9] key to point at two different objects and create a relationship (arrow) between them. These options are naturally not exclusive, therefore a retrieved knowledge-base can be updated with both English sentences and then activity cycle input.

Once a semantic network is partially built, it is important to be able to move about the network quickly in order to examine or change the semantic network. There is a single cursor which the user can move about the spreadsheet cells using the arrow keys (including [PAGEUP] and [PAGEDOWN]). Whatever the object the cursor is standing over is called the 'current object'. Pressing the [F2] key edits the name of the current object (e.g. to correct misspellings). In order to display all relationships for the current object, pressing the [F10] key displays them (these relationships could be on any of the levels). To see the other relationships graphically, pressing [PAGEUP] or [PAGEDOWN] keys changes the level to the next

level in which this object occurs. It is also important to be able to see an overview of the current level, and be able to 'zoom in' for an increased magnification of an area. This can be done by pressing the [CONTROL Z] key. The present implementation of zoom is limited to only one increased magnification of a problem (due to the poor graphical support available with the Turbo pascal language used to write the prototype)). It would be ideal if the resolution could be gradually increased or decreased by the user.

To aid experimentation (when adding or deleting objects or relationships), all new relationships added are displayed in a different colour (ie they are Non-permanent) until they are made permanent (by committing them) or are removed (ie Rolled Back). This is analogous to the Commit / Rollback option of commercial databases. Thus if a wrong sentence was typed in, its effects can be immediately and instantly unwound by using the '/ Worksheet Roll back' command in section 4.3.1. While pressing the [F3] key commits the changes done (identical to the '/ Worksheet Commit' command in section 4.3.1). Pressing the [DEL] key makes a permanent link, selected with the [SPACE] key, non-permanent, and thus enabling its deletion using 'roll back'.

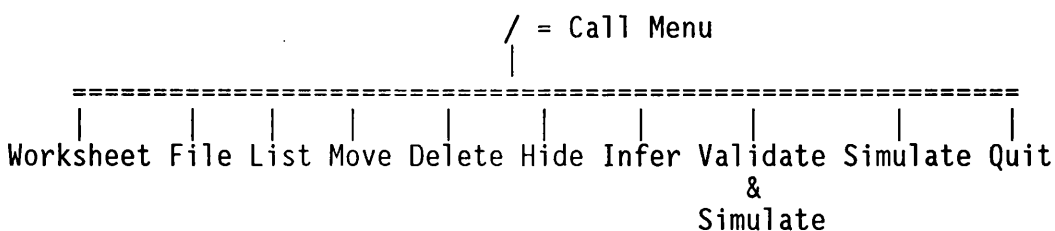
As on an ordinary spreadsheet, all objects and relationships are held in memory until the user saves the work into a file on disk by using the '/ File Save' command in section 4.3.2.

The user may wish to hide the borders and column/row numbers, for example to make the interface visually cleaner. This can be done by pressing the [CONTROL B] key.

Section 4.3 : MULTI LEVEL MENU SYSTEM

As mentioned in the previous section, the multi-level menu system, which is similar to the Lotus 1-2-3 menu interface, is accessed from the graphical interface by pressing the '/' key. The multi-level menu system is based on the concept that, at any one time, there are a number of options (including Sub-menus) which can be chosen by either moving the cursor over them and pressing [RETURN] or pressing the first letter of option title. When moving the cursor, a help line is shown at the line below the cursor to briefly explain the purpose of the highlighted option.

The top level menu structure is as follows :



The above sub-menus and are described in the next sections. Some of these options prompt for a range. The technique for highlighting a range are firstly to move the cursor to a corner of the range.

Then to press the full-stop to 'anchor' the cursor. To 'unanchor' the cursor press the [ESC] key. Thirdly to move the cursor to the opposite corner and press the [RETURN] key. If the full-stop key is pressed, the cursor is moved to the next corner in a clockwise direction.

Section 4.3.1 : Worksheet Sub-menu

This sub-menu is concerned with general worksheet settings. The following options can be chosen :

Reset : Clears all data in memory, thus prepares the worksheet for another session.

In : Zooms into the current cell of the worksheet. This has the effect of reducing the number of cells displayed at one time from 11 x 10 to 4 x 4. The column width would thus be set to 19.

Out : Zooms out. This returns the screen to display 11 x 10 cells. The column width would thus be set to 7.

Commit : This makes all permanent.

Roll Back : This removes all non-permanent relationships. If an object only had non-permanent relationships, then that object would be deleted.

Width : This sets the column width of the cells. It does not change the number of cells displayed vertically.

Section 4.3.2 : File Sub-menu

This sub-menu is concerned with loading and saving files from disk.

The following options can be chosen :

Retrieve : This clears all data in memory and retrieves a spreadsheet from the disk. A list of all spreadsheet files are displayed, and can be chosen by moving the cursor to the relevant spreadsheet and pressing return, alternatively the name of the file (including and change in the directory or drive) can be typed in.

Save : Save the worksheet in memory as a spreadsheet file on disk. The user will be prompted for the file name. The user can accept the suggestion by pressing [return], or type in the file name himself.

Combine : This combines the data in two spreadsheets. [Not yet implemented]

Translate : This takes english sentences from a file on disk, and translates it into the semantic network representation.

Extract : This extracts a range from the worksheet in memory and saves it as a separate spreadsheet file on disk. [Not yet implemented]

Section 4.3.3 : List Sub-menu

This sub-menu is concerned with listing objects and relationships in the present knowledge-base (spreadsheet). The following options can be chosen :

Objects : Lists objects in the worksheet, sorted alphabetically.

Relations : Lists relationships in worksheet, sorted alphabetically.

Links : Lists all links in the worksheet, not sorted.

Links_Relationships : This prompts for a relationship name, and it lists all links involving this relationship.

Spool On : This starts spooling the non-graphical screen output to a disk file. If the file already exists then all of its contents are erased.

Spool Stop : This stops spooling.

Section 4.3.4 : Move Sub-menu

This options allows the movement of objects from one place to another (on the same level). It prompts for a range to move, then it prompts for the top left hand corner of the range to move to. There is no danger of objects being 'over-written' since if a cell in the TO-RANGE is occupied, the corresponding cell in the FROM-RANGE is left in its old location.

Section 4.3.5 : Delete Sub-menu

This option deletes objects (and their links on all levels). It prompts for a range.

Section 4.3.6 : Hide Sub-menu

This removes objects from a level where that object does not have any links. If the object does not have any links at all, then the object is deleted. This option prompts for a range.

Section 4.3.7 : Infer Sub-menu

This is an option where SASIM tries to investigate the links between two objects (whether entities or activities). This might be useful if, after a simulation run, the results of a change in the number of one entity counter-intuitively affected another object. There are clearly many possible links, some of which are relevant, some of which may in certain conditions be relevant and some links which are blatantly wrong (an analogy of the complexity involved would be the

problem of finding every possible way to travel from A to B, with ample public and private transport, when money is no limitation and style and speed is irrelevant). The process will hopefully 'HIT' on a successful link (ie a link which turns out to be the causal link, analogous to the critical path in CPA analysis). The shortest links are highlighted first (based on the number of links).

The option first prompts for a causal object (chosen simply by using cursor keys), then it prompts for a second object which the user would like to have some insight into its impact if the first object was changed. The use would then be stepped through the first link chain found (stepping through by pressing the [SPACE] key). At the end of every link chain, the user has the option to abort this investigation, or to continue and locate another link chain.

The inference engine is based on the Branch and Bound algorithm. The inference engine is prevented from choosing objects 'ENTITY' or 'ACTIVITY', or relationships 'NUMBER' or 'DURATION' as part of the chain of relations since these are meaningless in this type of analysis, and it is also prevented from transversing the cycle of an entity the wrong way round.

This type of semantic analysis is only possible because the knowledge-base is limited to a simulated system. SASIM already knows the meaning of some relationships ('AFTER' or 'PRIORITY') as well as

some objects ('ENTITY' or 'ACTIVITY'). This is an essential constraint in this environment, but SASIM would need to be modified if it were to be used in another environment, such as medical diagnosis.

Section 4.3.8 : Validate Option

After the knowledge-base has been input, the software must provide a facility for validation (after all, there is a very high probability of the user missing out one or more relationships). This option tries to list cycle of every entity it has discovered, as well as listing the entities involved in every activities. It also checks that the cycles do not break into two or more separate components (in the PUB problem, described in section 6.2, it correctly notes that 'DOG' cycle is non-existent, as well as that it is not connected to other cycles).

The system will ask for any missing information necessary to run a simulation, including the starting points of entities, the stop time, the run in period and the pause interval. These will be stored on the NUMBER level of the semantic network. The next section will describe the menu option which starts the actual simulation run.

Section 4.3.9 : Simulation Run Option

This option starts a simulation run. The system will automatically ask for any missing information necessary to run a simulation (e.g. if the validate option in section 4.3.8 has not been run by the user). A description of the simulation run process is contained in section 4.6. If the user has the 'SPOOL TO FILE' option enabled, he will be asked whether he would like to store a full audit trail of the simulation run, or just a statistical summary.

Section 4.3.10 : Quit Sub-menu

This option allows the user to exit SASIM. **WARNING** If the worksheet has not been saved, it will be lost. SASIM prompts the user to confirm that he would like to continue and exit out of the software package.

Section 4.4 : NATURAL LANGUAGE UNDERSTANDING AND PROCESSING (NLUP)

The setting up of the semantic network manually, by entering each object in separate spreadsheet cells and then setting each relationship in turn (using the [F9] key), is very time consuming and error prone. There is therefore a need to enter an English sentence which could be translated into many semantic network objects and relationships. The interface which was developed in SASIM takes a free text english and converts it automatically into the semantic network format. This can clearly break the initial

barrier between the client and the simulation practitioner [Doukidis, 1985] in that the practitioner converts the client's dialog for the system in such a way that the user's understanding is retained. An example of a valid sentence is :

Customers after drinking, then rest for three minutes.

(all sentences finishing with a full-stop)

This uses the syntax :

ENTITY AFTER ACTIVITY THEN ACTIVITY {ACTIVITY_DESC}.

The syntactic formats allowed are summarised in Appendix 1. This syntax can be expanded by simply changing a data file. Appendix 3.1 contains a full NLUP worked example for the PUB problem.

In SASIM, these sentences can be either typed in, one after another, using function key [F8], or alternatively by first creating a file using any text editor, such as Turbo Pascal or Wordstar (using non-document mode) or Wordstar 2000 (using format "unformatted"). The only constraints on the file are that the lines are not longer than 100 words, and the maximum length of a word is 49 letters. As a convention, the file should have an extension of '.ENG'. To read this file, the option from the menu system is from the 'FILE' Sub-menu, Option 'TRANSLATE'.

The problem of different words meaning the same thing (for example Service and Serving), as well as singular and plural representations of words, has been solved by using a 'dictionary'. This provides a 'mask' approach which changes all nouns to the singular, as well as all verbs to a common tense (for example Serving to Service). Appendix 3 lists these masks, which are checked in turn, starting from the first one on the list, and as soon as one mask is found to fit, the conversion is stopped. All inputs are also converted to upper case. The problem of synonyms has also been handled by a second dictionary look up.

A very useful feature is the automatic handling of arrival (or regular occurring) mechanisms. It looks for the keyword 'EVERY', and it then builds in an arrival (regulating) mechanism.

SASIM attempts to tackle the translation from the written sentence, but not the spoken word, although if a speech subsystem was used it could be easily integrated into SASIM.

Section 4.5 : ACTIVITY AND ENTITY CYCLE INPUT INTERFACE

In addition to the NLUP interface, SASIM provides a graphical interface to quickly input either ACDs, whether partially or fully completed, or simply a list of activities and/or entities. These would be automatically translated into the semantic network

knowledge-base, and would thus enable the integration of both NLUP and graphical cycle input to be used in the formulation stage of a simulation.

The input process is started by pressing the [F6] key. SASIM would then prompt for an entity and/or an activity, which can be chosen by just typing in its name, or if the object is already defined, using the arrow keys to locate the object on the semantic network and pressing <enter>. If SASIM does not already know this object is an entity or an activity it would prompt the user to indicate which one it is.

The interface is based around the concept of two state counters, one indicating the current instance of an entity, the other indicating the current activity. These two counters are indicated at the bottom of the screen.

If the user chooses an activity then the current instance is assumed to move from the present instance to the new activity. If the user chooses an entity (or any of its instances), then the entity is assumed to be involved in current activity. In both cases, SASIM notifies the user of all links added to the semantic network. To exit, press the [ESC] key at any stage.

The following is an example of using the Activity and Entity Cycle Input interface. Starting with an empty semantic network knowledge-base, the following is a typical user session :

user Input =====	SASIM reply =====
[F6] key	Please input Activity and/or Entity
CUSTOMER [enter]	'ACTIVITY or ENTITY' menu
{choosing} ENTITY	'CUSTOMER IS A ENTITY' added, press any key to continue
{any key}	Please input Activity and/or Entity
ARRIVE {enter}	'ACTIVITY or ENTITY' menu
{choosing} ACTIVITY	'ARRIVE IS A ACTIVITY' added, press any key to continue
{any key}	'CUSTOMER (ARRIVE) IS A CUSTOMER' added press any key to continue
{any key}	'ARRIVE NEEDS CUSTOMER (ARRIVE)' added press any key to continue
{any key}	Please input Activity and/or Entity
SERVED	ACTIVITY or ENTITY' menu
{choosing} ACTIVITY	'SERVICE IS A ACTIVITY' added, press any key to continue
{any key}	'CUSTOMER (SERVICE) IS A CUSTOMER' added press any key to continue
{any key}	'SERVICE NEEDS CUSTOMER (SERVICE)' added press any key to continue
{any key}	'CUSTOMER (ARRIVE) PRECEDES CUSTOMER (SERVICE)' added press any key to continue
{any key}	Please input Activity and/or Entity
BARMAID	'ACTIVITY or ENTITY' menu
{choosing} ENTITY	'BARMAID IS A ENTITY' added, press any key to continue
{any key}	'BARMAID (SERVICE) IS A BARMAID' added press any key to continue
{any key}	'SERVICE NEEDS BARMAID (SERVICE)' added press any key to continue
{any key}	Please input Activity and/or Entity
[ESC]	returns to normal view.

Section 4.6 : SIMULATION RUNNING PROCESS

The basic concept behind the simulation running process is that the user still sees the original semantic network while seeing each entity involved in an activity (or moving between activities). For example CUSTOMER (DRINK), has two counters: the counter above displays the number of occurrences waiting to start that activity, and the counter below displays the number of occurrences actually involved in that activity. By pressing the [CONTROL H] key, the user can view the histograms of an entity, including the queuing time and queuing length of that entity, as described in section 2.4. When an occurrence of an entity moves from one activity to another, the user actually sees the icons moving simultaneously on the screen. In addition, the text area at the top of the screen indicate the name of present object, the status of the simulation (ie paused, running or stopped), the time in the simulation and the simulation delay. The user can change this delay factor by pressing the '<' or '>' key. Pressing the [SPACE] key pauses or recontinues the simulation, which would be useful if the user wished to examine the state of the system during the simulation run. Pressing the [ESC] key stops the simulation run.

Section 4.7 : CONCLUSION

This chapter has presented SASIM, a prototype implementation of the semantic modelling approach. The prototype uses the familiar Lotus 1-2-3 menus and spreadsheet concept to reduce the conceptual jump required for many user. The prototype implements all the semantic modelling concepts, including natural language processing and simulation running. As discussed in section 3.7, complex problems require additional conditions for cooperation between objects. The implementation of this procedural attachment concept in the prototype is described in the next chapter.

CHAPTER 5

PROCEDURAL ATTACHMENT

Section 5.1 : INTRODUCTION

Section 3.6 highlighted the advantages of procedural attachment as a means to represent complex cooperation between objects. This chapter presents the implementation of procedural attachment in the SASIM prototype.

SASIM implements procedural attachment of active production rules (written in Turbo Pascal) into the simulation process, thus taking charge of selected sections of semantic network (localized control). There is the ability to set up and monitor attributes, and to set up peculiar restrictions and interactions which can not be easily represented in a semantic network. The decision to use active production rules, as opposed to passive production rules, was taken because of the complexities of writing the interpreter required for passive production rules (whereas active production rules are compiled by the Turbo Pascal compiler). Unfortunately, the inference engine is not able to analyse active production rules. Despite their less flexible nature, active production rules illustrate the use of procedural attachment to supplement the semantic network knowledge-base.

It is important to realise that the user supplied code, whether input directly by the user or aided by the in built 'PROGRAM GENERATION' option, does not create a completely different environment as is customary in other environments. What it does is to create a customised SASIM environment which interacts directly with the latest semantic network knowledge-base and modifies the movement and cooperation between entities in selected areas of the semantic network during simulation running. In other words, procedural attachment enables selected activities to 'opt out' of the in-built simulation running executive and thus be controlled and managed by the user supplied code, with reference still to the graphical semantic network knowledge-base. This still enables modification of the semantic network AFTER the user code is created or modified.

This chapter first details the structure of the procedural attachment program file (named USERCODE.PAS). All user supplied code is contained in this file. Section 5.5 details the main callable procedures defined by SASIM. These would be mixed with standard Turbo Pascal code in order to define the required relationships. The user supplied code file, USERCODE.PAS, needs to be compiled by Turbo Pascal which automatically integrates the code into the simulation environment, creating a customised internal knowledge-base (contained in the main executable file SASIM.EXE). Appendix 3.6 illustrates the use of procedural attachment in the pub problem as described in section 6.2.

Section 5.2 : STANDARD TERMINOLOGY

The idea that an **entity**, such as a **CUSTOMER**, has multiple **instances**, for example **CUSTOMER (SERVICE)** or **CUSTOMER (DRINK)** has already been described. Since each instance is a different state which the entity can be in, relationships between the instances can be defined on the semantic network, for example "**CUSTOMER (SERVICE)**" proceeds "**CUSTOMER (DRINK)**". However when entities interact, it is important to both keep track of different occurrences of a single instance, and occurrences of an entity when it moves from one instance to the next.

For example, to monitor the time that a specific customer takes from entering the pub to leaving it, the customer is allocated a unique occurrence code which he carries with him and which defines other data specific to him (for example arrival time, number of drinks so far etc.).

Entities, instances, activities or descriptions are **objects** since they are located at discrete cells on the semantic network. The actual location of the objects on the semantic network is irrelevant to the user code.

Section 5.3 : STRUCTURE OF PASCAL USER CODE

All user code is in a file called 'USERCODE.PAS'. This file has separate high-level sections (or procedures), each assigned a different role. Since these procedures are called by SASIM during a simulation, they can directly affect the course of a simulation. These high-level procedures can contain any Turbo Pascal procedures or specialised SASIM callable procedures (highlighted in section 5.4). If the high-level procedures do not contain customised code, or only contain code for selected objects, the unmentioned objects contained in the knowledge-base are not affected, thus achieving the goal of 'attaching' code to objects.

The first high-level section is the occurrence definition section. For example, the pub problem requires the creation an integer data item for each customer to log the number of drinks required. The pub problem contained the following occurrence definition section :

```
=====
{$I USERCODE.IN1}
  ATT = RECORD
                INTERNAL_USE : OCC;
                NO_DRINKS : INTEGER;
  END;
{$I USERCODE.IN2}
=====
```

The second high-level section is the global variable definition section. For example, to create a histogram requires the definition of a pointer to a predefined histogram record (called HHISTOGRAM). It is also desirable to be able to define variables to point to

selected objects on the semantic network (OOBJECT is a predefined pointer to an object). The pub problem contained the following global variable definition section :

```
=====
VAR
  DOOR_ARRIVE, DOOR_ARRIVE_ARR,
  CUSTOMER_ARR, ARRIVE, CUSTOMER_REST, CUSTOMER_SERVICE,
  CUSTOMER_EXIT, GLASSES, CUSTOMERS, BARMAIDS,
  GLASS_WASH : OOBJECT;
  DRINK_HIST, SERVE_TS, BARMAID_TS : HHISTOGRAM;
=====
```

The third high-level section is the set hooks procedure. This procedure is called once at the beginning of each simulation run to enable 'hooks' to be inserted into the semantic network for all objects which are to be managed by this pascal code. These hooks are latched on with the LOCATE_OBJECT function. Once latched on, an activity can be set for manual (customised code) control by setting the CONTROL field in the object to USER (e.g. ARRIVE^.CONTROL := USER). This section would also be used to define customised histograms (using the INITIALISE_HIST procedure defined in section 5.4) - noting that all instances already have, by default, a queuing time and length histogram defined. This section is also ideal to contain the ICON_MATRIX procedure (defined in section 5.4) which defines the graphical shape of objects using a simple 2x2 character

matrix. The pub problem contained the following set hooks procedure:

```

=====
PROCEDURE SET_HOOKS;
BEGIN
  DOOR_ARRIVE := LOCATE_OBJECT('DOOR_ARRIVE');
  DOOR_ARRIVE_ARR := LOCATE_OBJECT('DOOR_ARRIVE (ARRIVE)');
  CUSTOMER_ARR := LOCATE_OBJECT('CUSTOMER (ARRIVE)');
  CUSTOMER_REST := LOCATE_OBJECT('CUSTOMER (REST)');
  CUSTOMER_SERVICE := LOCATE_OBJECT('CUSTOMER (SERVICE)');
  CUSTOMER_EXIT := LOCATE_OBJECT('CUSTOMER (EXIT)');
  CUSTOMERS := LOCATE_OBJECT('CUSTOMER');
  GLASSES := LOCATE_OBJECT('GLASS');
  GLASS_WASH := LOCATE_OBJECT('GLASS (WASH)');
  BARMAIDS := LOCATE_OBJECT('BARMAID');
  ARRIVE := LOCATE_OBJECT('ARRIVE');
  IF NOT ERR THEN
    ARRIVE^.CONTROL := USER;
  ICON_MATRIX(0,0,238,221,154,WHITE,BLACK,GLASSES); (*ε 238,221
                                                    Ü 154*)
  ICON_MATRIX(0,0,12,11,1,YELLOW,BLACK,CUSTOMERS); (* Face *)
  DEFINE_CONVEYOR(-1,0,1,0, 2, GREEN, GLASS_WASH); (* X,Y, XSTEP,
                                                    YSTEP, LENGTH, COLOUR, INSTANCE *)
  INITIALISE_HIST(DRINK_HIST,CUSTOMER_ARR,'NUMBER OF DRINKS');
  INITIALISE_TSERIES(SERVE_TS,CUSTOMER_SERVICE,
                    'CUSTSERV',3,0,15);
  INITIALISE_TSERIES(BARMAID_TS,BARMAIDS,'BARMAID',3,0,15);
END;
=====

```

The fourth high-level section is the graphic object procedure. This procedure would be used to define complex graphical shapes (using Turbo Pascal graphics). This procedure is called for every object on the network just before it's name is output on the screen. This object can then, if desired, be drawn graphically rather than just named on the screen. The procedure (function) returns a number, being the length, in character positions, of the name (to accurately locate the start and end of the relationship arrows). If the number

returned is negative, this would indicate that the object has been drawn graphically, and therefore does not require its name to be output. The pub problem contained the following graphic object procedure:

```

=====
FUNCTION GRAPHIC_OBJECT(OBJECT : OBJECT; XG1,YG1 : INTEGER;
                        DRAW : BOOLEAN) : INTEGER;
VAR
  T1,T2 : INTEGER;
  PENTAGON : ARRAY [1..4] OF POINTTYPE;
BEGIN
  IF OBJ^.NAME = 'CUSTOMER (DRINK)' THEN
  BEGIN (* shape of a man *)
    IF DRAW THEN
    BEGIN
      SETCOLOR(WHITE);
      YG1 := YG1-YDOTS_PER_CHAR;
      XG1 := XG1+XDOTS_PER_CHAR;
      CIRCLE(XG1,YG1,ROUND(YDOTS_PER_CHAR*0.2));
      YG1 := YG1+ROUND(YDOTS_PER_CHAR*0.2);
      LINE(XG1,YG1,XG1,YG1+YDOTS_PER_CHAR);
      YG1 := YG1+YDOTS_PER_CHAR;
      LINE(XG1-ROUND(XDOTS_PER_CHAR*0.5),
           YG1+ROUND(YDOTS_PER_CHAR*0.5),XG1,YG1);
      LINE(XG1+ROUND(XDOTS_PER_CHAR*0.5),
           YG1+ROUND(YDOTS_PER_CHAR*0.5),XG1,YG1);
      YG1 := YG1-ROUND(0.66*YDOTS_PER_CHAR);
      LINE(XG1-ROUND(XDOTS_PER_CHAR*0.5),
           YG1,XG1+ROUND(XDOTS_PER_CHAR*0.5),YG1);
    END;
    GRAPHIC_OBJECT := -3;
  END
  ELSE
  IF OBJ^.NAME = 'CUSTOMER (SERVICE)' THEN
  BEGIN (* money sign to indicate payment when served *)
    IF DRAW THEN
    BEGIN
      SETCOLOR(WHITE);
      RECTANGLE(XG1-13,YG1-4,XG1+21,YG1+12);
      SETUSERCHARSIZE(5,5,5,5);
      SETTEXTSTYLE(SMALLFONT,HORIZDIR,USERCHARSIZE);
      SETCOLOR(GREEN);
      OUTTEXTXY(XG1-10,YG1,'MONEY');
    END;
    GRAPHIC_OBJECT := -5;
  END
  ELSE

```



```

IF (OBJ^.NAME = 'GLASS (SERVICE)') OR
   (OBJ^.NAME = 'GLASS (DRINK)') THEN
BEGIN   (* beer glass shape *)
  IF DRAW THEN
  BEGIN
    SETCOLOR(BROWN);
    setfillstyle(solidfill,BROWN);
    PENTAGON[1].X := XG1+2;
    PENTAGON[1].Y := YG1+ROUND(YDOTS_PER_CHAR*0.5);
    PENTAGON[2].X := XG1+2*XDOTS_PER_CHAR-2;
    PENTAGON[2].Y := YG1+ROUND(YDOTS_PER_CHAR*0.5);
    PENTAGON[3].X := XG1+2*XDOTS_PER_CHAR;
    PENTAGON[3].Y := YG1-YDOTS_PER_CHAR;
    PENTAGON[4].X := XG1;
    PENTAGON[4].Y := YG1-YDOTS_PER_CHAR;
    DRAWPOLY(SIZEOF(PENTAGON) DIV
              SIZEOF(POINTTYPE),PENTAGON);
    FILLPOLY(SIZEOF(PENTAGON) DIV
              SIZEOF(POINTTYPE),PENTAGON);
    PIESLICE(XG1,YG1-YDOTS_PER_CHAR,90,270,2);
    PIESLICE(XG1+2*XDOTS_PER_CHAR,
              YG1-YDOTS_PER_CHAR,0,90,2);
    PIESLICE(XG1+2*XDOTS_PER_CHAR,
              YG1-YDOTS_PER_CHAR,270,360,2);
    SETCOLOR(WHITE);
    setfillstyle(INTERLEAVEFILL,WHITE);
    BAR(XG1,YG1-ROUND(1.2*YDOTS_PER_CHAR),
         XG1+2*XDOTS_PER_CHAR,YG1-YDOTS_PER_CHAR);
  END;
  GRAPHIC_OBJECT := -2;
END
ELSE
  IF OBJ^.FORGROUND_COLOUR = -1 THEN   (* default section *)
    GRAPHIC_OBJECT := LENGTH(OBJ^.NAME)
  ELSE
    GRAPHIC_OBJECT := 2;
END;

```

=====

The fifth high-level section is the B event procedure. This procedure is used to define where an occurrence, OCCUR, moves on to after completion of the activity associated with instance CURRENT_OBJECT (i.e. during the B phase in a three phase simulation). If the movement of an occurrence is not explicitly defined, it is automatically moved on according to the semantic network knowledge-base. The pub problem contained the following B event procedure :

```

=====
PROCEDURE B_EVENT(CURRENT_OBJECT : OOBJECT; OCCUR : OOC);
BEGIN
  IF CURRENT_OBJECT = CUSTOMER_REST THEN
    BEGIN
      DEC(ATTRIBUTE(OCCUR)^.NO_DRINKS);
      IF ATTRIBUTE(OCCUR)^.NO_DRINKS > 0 THEN
        ADD_TO_B_QUEUE(CUSTOMER_SERVICE, OCCUR)
      ELSE
        ADD_TO_B_QUEUE(CUSTOMER_EXIT, OCCUR);
    END
  END;
=====

```

The sixth high-level section is the C event procedure. This procedure is used to define when activities begin (i.e. during the C phase in a three phase simulation). This procedure is only called for activities which are set for manual (customised code) control in the set hooks procedure above. The pub problem contained the

following C event procedure :

```
=====
PROCEDURE C_EVENT(CURRENT_ACTIVITY : OOBJECT);
VAR
  SAMPLE_TIME : INTEGER;
  OCCUR : OCC;
BEGIN
  IF CURRENT_ACTIVITY = ARRIVE THEN
    WHILE (QSIZE(DOOR_ARRIVE) > 0) AND (QSIZE(CUSTOMER_ARR) > 0)
    DO
      BEGIN
        SAMPLE_TIME := SAMPLE_ACT_TIME(ARRIVE);
        OCCUR := BEHEAD(CUSTOMER_ARR);
        ATTRIBUTE(OCCUR)^.NO_DRINKS := RANDOM(3)+2;
        UPDATE_HIST(DRINK_HIST,ATTRIBUTE(OCCUR)^.NO_DRINKS,1);
        ADD_TO_TREE(SAMPLE_TIME,OCCUR,CUSTOMER_ARR);
        ADD_TO_TREE(SAMPLE_TIME,BEHEAD(DOOR_ARRIVE),
                    DOOR_ARRIVE_ARR);
      END;
    END;
  END;
=====
```

The seventh high-level section is the report procedure. This procedure is called at the end of each simulation run (or sub-run), and is thus the place to dictate which histograms to display. The pub problem contained the following report procedure :

```
=====
PROCEDURE REPORTS;
BEGIN
  DISPLAY_HISTOGRAM(DRINK_HIST);
  PRINT_HIST(CUSTOMER_SERVICE);
END;
=====
```

The eighth high-level section is the main section. This procedure is called only once on loading the SASIM prototype. It could be used to initialise some global initialisation. However its use is not required in the pub problem. The pub problem thus contained the

following default main section :

```
=====
BEGIN
  {BLANK MAIN}
END.
=====
```

Section 5.4 : CALLABLE PROCEDURES

The section details the main procedures defined by SASIM which can be called by the user supplied code. These would be mixed with standard Turbo Pascal code in order to define the required relationships. The defined procedures can be split into groups according to which of the eight sections they are most likely to be used in. To understand the parameters required, the following type definitions are used :

HHISTOGRAM is a pointer to a histogram.

LSTRING is defined as a standard string.

OCC is a pointer to an occurrence record.

OBJECT is a pointer to an object record. This can be any object on a semantic network, including an instance or an activity.

The third high-level section, the set hooks procedure, may require the following procedures :

FUNCTION LOCATE_OBJECT(Obj_Name : LSTRING) : OOBJECT;

Returns the pointer to an object (whether an activity, entity, instance or description) whose name is "Obj_Name". This function thus provides the hooks in the semantic network through which all the defined functions are dependent upon. This procedure would be placed in the SET_HOOKS section, since it only needs to be called once.

PROCEDURE ICON_MATRIX(TopLeft, TopRight, BottomLeft, BottomRight, Character, ForegroundColor, BackgroundColor : INTEGER; Obj : OOBJECT);

Defines the object "Obj" on the screen using a simple 2x2 character matrix. The ASCII of each character in the matrix is passed in the function call. In addition an ASCII character for each occurrence to indicate an occurrence moving from one instance to another is also passed as parameter "Character". The background and foreground colours of the object can be specified. An ASCII of '0' indicates a default value. This procedure would be placed in the SET_HOOKS section, since it only needs to be called once.

```
PROCEDURE INITIALISE_HIST( Hist : HHISTOGRAM; Obj : OBJECT;  
                          Hist_name : LSTRING );
```

This creates a customised histogram "Hist" whose name is "HIST_NAME", and attaches it to object "Obj". It is the user code's responsibility to update a customised histogram. This procedure would be placed in the SET_HOOKS section, since it only needs to be called once.

The fifth high-level section, the B event procedure, may require the following procedures :

```
PROCEDURE ADD_TO_F_QUEUE( Obj : OBJECT; Occur : OCCC);
```

```
PROCEDURE ADD_TO_B_QUEUE( Obj : OBJECT; Occur : OCCC);
```

Adds an occurrence "Occur" either to the front or the back of the associated with instance (or facility) "Obj". This would be used in the end phase of the three-phase simulation (i.e. B-phase) to move an occurrence forward to another queue.

```
FUNCTION RETURN_NUMBER( Occur : OCCC ) : INTEGER;
```

Returns a unique number for occurrence "Occur". This is a unique sequential number for all occurrences of an entity. This would be required if it is desired to trace individual occurrences through the system. This may also be useful in the C event procedure.

FUNCTION RETURN_UNIQUE_NUMBER(Occur : OOC) : INTEGER;

Returns a unique number for occurrence "Occur". This is a unique sequential number for all occurrences regardless of the actual entity. This would be required if it is desired to trace individual occurrences through the system, possibly when occurrences 'transform' from one entity to another. This may also be useful in the C event procedure.

PROCEDURE UPDATE_HIST(Hist : HHISTOGRAM; Obs,Occ : INTEGER);

This updates histogram "Hist" with an observation, "Obs", which occurred "Occ" times. This is required to build up the statistical data in the histogram (for example the number of drinks). This may also be useful in the C event procedure.

The sixth high-level section, the C event procedure, may require the following procedures :

FUNCTION QSIZE(Obj : OOBJECT) : INTEGER;

Returns the size of the queue associated with instance (or facility) "Obj". This function would be used to find out if there are any occurrences of an entity waiting to be involved in an activity.

FUNCTION CYCLE(Obj : OOBJECT) : OOCC;

Causes the front occurrence of the queue associated with object "Obj" to be sent to the back, and returns the occurrence now at the front of the queue. This function would be used to scan all occurrences in a queue (for example to find out who should be served first).

FUNCTION SAMPLE_ACT_TIME(Current_activity : OOBJECT) : INTEGER;

Samples a time of activity "Current_activity", based upon the information in the semantic network. This would be used in the C_EVENT section when it is desired to scan all occurrences in a queue (for example to find out who should be served first).

FUNCTION RND(S:STREAM):REAL;
FUNCTION NORMAL(M:REAL; SD:REAL; S:STREAM):INTEGER;
FUNCTION NEGEXP(M:REAL; S:STREAM):INTEGER;
FUNCTION WEIBULL(A,B:REAL; S:INTEGER):INTEGER;
FUNCTION POISSON(M:REAL; S:INTEGER):INTEGER;
FUNCTION ERLANG(EI:INTEGER; M:REAL; S:STREAM):INTEGER;

These are standard sampling functions. This would be placed in the C_EVENT section to sample durations, attributes etc.

FUNCTION BEHEAD(Obj : OOBJECT): OOCC;

Returns the occurrence at the front of the queue associated with instance (or facility) "Obj". This function is generally called as a prelude to calling the ADD_TO_TREE procedure - see next.

PROCEDURE ADD_TO_TREE(Duration : INTEGER; Occur : OOC;

Instance : OOBJECT);

Starts an occurrence "Occur" into the activity associated with instance "Instance". The duration of the activity is "Duration". This would be used in the start phase of the three-phase simulation (i.e. C-phase) to indicate the start (and thus end) of an activity.

The seventh high-level section, the report procedure, may require the following procedures :

PROCEDURE DISPLAY_HISTOGRAM(Hist:HHISTOGRAM);

This displays on the screen histogram "Hist". This should only be used in the REPORTS section. If spool is switched on, it will print to a file as well.

PROCEDURE PRINT_HIST(Obj : OOBJECT);

Prints all histograms of object "Obj". This should only be used in the REPORTS section. If spool is switched on, it will print to a file as well.

Section 5.5 : COMPILING USER CODE

To compile the code, simply use the TURBO 4 compiler, loading up the file SASIM.PAS and pressing [F9] to compile the code. This is now a customised version of SASIM.EXE (the original version should be kept safe), and can be run as normal.

Section 5.6 : AUTO GENERATING USER CODE

To speed up the customization process (using procedural attachment), SASIM can create a sample USERCODE which contains the HOOKS onto the Semantic network for the objects which the user wishes to modify.

As indicated above in section 4.5, the user is prompted as to whether to generate the user code after each simulation run. If the user chooses to generate the user code, the user is prompted to indicate all objects he is interested in, by using the cursor keys and pressing {enter} for each object in turn, and then press {esc}.

It is again important to restate that program generation does not mean, as in traditional systems, creating code describing the whole problem. What program generation does is to speed up the creation of the hooks onto selected areas of the semantic network (i.e. calling the LOCATE_OBJECT() function) and setting up the 'C' event code for selected activities.

Section 5.7 : CONCLUSION

This chapter has presented the implementation of procedural attachment in SASIM. The decision to use active production rules, as opposed to passive production rules, was taken for pragmatic purposes. Despite their less flexible nature, active production rules illustrate the use of procedural attachment to supplement the semantic network knowledge-base and thus modify the running of a simulation.

An illustration of the use of procedural attachment in the pub example is contained in appendix 3.6. The next chapter critically analyses both the semantic modelling approach and the prototype.

CHAPTER 6

ANALYSIS OF SEMANTIC MODELLING

Section 6.1 : INTRODUCTION

The previous chapters have presented both the semantic modelling approach and the SASIM prototype implementation of this approach. This chapter presents two case studies where SASIM was used to implement a simulation. The first case study is the implementation of the simple pub example. The second is a more complicated problem on the effect of warship and replenishment ship attrition on war arsenal requirements. These case studies will provide vital feedback on the practicality of implementing the semantic modelling approach and provide the foundation for future improvements and research. Section 6.4 will present an analysis of the semantic modelling approach.

Section 6.2 : THE PUB EXAMPLE

The pub example is a common first simulation written by newcomers learning about simulation. For this reason, it was chosen as the first test of SASIM. The detailed description of the pub example and the output from the SASIM prototype is contained in appendix 3. The standard simulation does not contain the entity Dog, however I have added it to this case study in order to illustrate the validate option's ability to spot entities which are not connected to other entities.

Since the example is commonly described in english, it was natural to use the natural language interface as the main technique for inputting the description. This simple problem was useful since initially the syntactic structures and dictionary were very basic. It highlighted the requirement to cope with different words meaning the same thing (for example SERVED refers to SERVICE etc). These experimental improvements are expected in a prototype system, and similar enhancements will be required when coping with other domains. There were also a few phrases in the initial description of the pub which needed to be re-phrased before inputting in to the natural language interface. This was due to either their ambiguity or superfluousness. The alternative of adding the syntactic structure to the natural language interface was done in most cases.

Each sentence was translated in turn by the natural language interface, in each case listing the sentence as well as the translation. These are listed in appendix 3. There were 13 sentences in the original english description. These were translated to 107 semantic network links (appendix 3.2). Having translated the sentences, the semantic network looked untidy on the screen. Some time was therefore necessary to move the objects to provide a neater view. The tidied up semantic network can be seen in appendix 3.7.

It was clear from the entity cycle level that the customer cycle was not complete. There was no link between CUSTOMER (EXIT) and CUSTOMER (ARRIVE). This link is not defined by the english but is just a convention used in simulation, in order to close all entity cycle loops. The link "GLASS (SERVICE) PRECEDES GLASS (DRINK)" was also missing. Even though it is possible to spot and add these two links manually, the validate section automatically located these missing links and added them, assuming all that was missing was a single link to close the loop (as can be seen from the screen output of the validate option in Appendix 3.5.2). Since it is possible that the validate option could be run before the whole system is described, the system should have prompted the user as to whether to add the 'missing' links or not. Various missing data was also prompted for, including the number of glasses and the duration of activity Exit. The validate section also highlighted a number of

anomalies, notably that the condition "IF THEY DRUNK LESS THAN THREE TIMES" can not be directly interpreted because the prototype does not provide automatic support of attributes, since a full interpretation requires the procedural attachment modifications shown in appendix 3.6. So as to provide a partial interpretation during a simulation run, the probability of any customer satisfying this condition is requested. The probability is set at 66.6% (ie that a customer has a 66.6% chance of ordering another drink after finishing one glass, and a 33.3% probability of exiting). The validate section also highlighted that there was no cycle for entity DOG and it was not linked to the rest of the network. The validate section listed the cycles of all the entities, as well as listing all entities involved in each activity.

Procedural attachment was also used to define graphical icons for the objects (for example glasses), as well as adding a conveyor belt for the glass washing activity. This is most notable in the top graphical level (shown in appendix 3.7). The full listing of the procedural attachment code is in appendix 3.6.

The simulation was run on the enhanced semantic network, with procedural attachment code. Various time series, histogram of queue length and queuing times are shown in appendix 3.7. The simulation subsystem in SASIM provided the ability to examine the behaviour of the system both during the running of the simulation and at preset

intervals defined before the start of the simulation. A detailed printout of all activity start and end timings is possible for validation and verification purposes.

Section 6.3 : WAR ARSENAL PROBLEM

The problem "The Effect of Warship and Replenishment Ship Attrition on War Arsenal Requirements" is described by Holder and Gittins (1989). This is a real world problem which was solved using the eLSE simulation subsystem [Crookes et al, 1986]. There was a military requirement to assess whether the stores held by warships and replenishment ships is presently set at an appropriate level once the possibility of destruction of the ships by enemy action has been taken into account. The basic set up is that ships are in one of three layers, the outer, inner or core layers. They could be attacked by either aircrafts or submarines, where they must engage outer ships first, then if they survive they may engage inner ships and lastly core ships. These layers create a repetitive type of activity cycle.

Even though, because of the military sensitivity of this kind of simulation, it was not possible to obtain detailed data on the problem, this case study was chosen because of the complexity of the cycles and its repetitive nature which would require a large semantic network, thus stretching the SASIM prototype. It was not

the intention to actually run a simulation, but to create a knowledge-base containing the main activity cycles.

The main means to enter the problem was the "Activity and Entity Cycle Input". This was chosen since an activity cycle diagram for the problem was given. The natural language interface was most useful for arrival mechanism entry. By simply saying "x arrives every y minutes", SASIM creates a door mechanism (where an entity arrives, based on a statistical distribution, from an indefinitely large pool of entities). The considerable size of the network presented a few memory and location problems. For example there are 25 activities, each having a link with the object "ACTIVITY". It is not easy to show all these links on a small PC screen, while still maintaining legibility. The full semantic network can be seen in appendix 4.

Building the large semantic network was very beneficial in understanding the war arsenal problem (especially when examining the entity cycle level of the semantic network). However, this understanding, gained by both the simulation practitioner and the client, could also be a side effect of any kind of modelling exercise.

Building the large semantic network highlighted some limitations of SASIM's graphical interface, which will be analysed in the next

section. By the nature of the problem, procedural attachment is required to provide a fully working model of the system, but due to the military sensitivity of the problem, not enough details of the problem were available.

Section 6.4 : ANALYSIS OF SEMANTIC MODELLING

One of the initial reasons for choosing semantic networks as the central knowledge-base is its graphical nature. It should thus be theoretically easier to spot errors and misunderstandings than in textual databases. An example of one such error is when the semantic network breaks down into more than one part due to a missing link. This beneficial effect was indeed noticed in the prototype, but not without noting that the screen displays (and graphical software) used is not as powerful as ideally required. This short coming is most noticed in the lack of clarity on the screen of some of the object names. This was because the small fonts needed a higher resolution monitor than the IBM PS/2's VGA screen. A larger screen would be advantageous since it would enable the display of more of the semantic network at any one time. With a higher resolution display, the option of seeing each entity in a queue and not just the grand total of the numbers in a queue may be achievable. Icons should also be able to change shape depending on which activity or queue they are involved in. Defining objects using an icon editor would also simplify the interface. The above

can best be achieved by moving to a far more powerful graphical sub-system, for example Windows or Macintosh. This graphical sub-system should accept a mouse, thus relationships could be drawn by just pointing and clicking.

The natural language interface's syntactic structures and dictionary could also be made far more thorough. The present structures have been built up on a case by case basis. A far more thorough investigation of syntactic structures would be very beneficial. It would also be desirable for the system to learn new structures automatically (learning by example).

It was also noted that implementing procedural attachment of passive production rules, using the Turbo Pascal compiler, considerably steepened the learning curve for complex simulations. Additionally, these production rules were not used in the inference engine. Both these problems could possibly be overcome by making the production rules passive (i.e. textual).

There is also a class of changes which would make the simulation running more powerful. Seed numbers could be methodically tackled, rather than allocated randomly in the validate section. Statistical analysis could also be improved, together with more advanced statistical sampling. The possibility of relationships being 'weighted', whereby a high weight would indicate a distant

relationship, should be analysed to see whether it could aid logical inferencing.

Section 6.5 : CONCLUSION

This chapter has presented an analysis of the practical implications when using semantic modelling in its present implementation, which will be very important in developing the next prototype. Section 3.11 described how semantic modelling tries to achieve most of the requirements of an ideal simulation support environment proposed in section 2.4. Therefore the main question is whether semantic modelling is implementable.

It is clear that the SASIM prototype is not ready for commercial use. For this, it would require a number of the improvements highlighted in section 6.4 to be implemented, especially higher definition screens and the implementation of passive production rules. However, even though SASIM has many rough edges and can be made easier to use, quicker to run, and can be updated to support far bigger simulations, its most important contribution is that it has shown the potential of semantic modelling through its implementation.

CHAPTER 7

SUMMARY AND CONCLUSION

Section 7.1 : SUMMARY

The objective of this thesis was to examine the use of artificial intelligence in the discrete event simulation field with the aim of examining some potential areas in which it might be possible to improve simulation environments. To this end, Chapter 1 described the general discrete event simulation environment, including the stages in the traditional simulation process. Chapter 2 presented some of the current research in the use of artificial intelligence in simulation and speculated what an ideal simulation environment should do. Chapter 3 demonstrated semantic modelling, a discrete event simulation modelling approach based on semantic networks, which attempts to give a consistent graphical interface throughout the life cycle of a simulation study, and described how semantic modelling tries to achieve most of the requirements of an ideal simulation support environment. Chapter 4 and 5 described the prototype implementation of the semantic modelling approach. Chapter 6 critically analysed both the semantic modelling approach and the prototype.

Section 7.2 : CONCLUSION

Existing simulation research in the artificial intelligence (AI) field is extended by investigating the graphical AI knowledge-base called semantic networks. This thesis has demonstrated semantic modelling, a discrete event simulation modelling approach based on semantic networks, which attempts to give a consistent graphical interface throughout the life cycle of a simulation study. The semantic modelling approach is an extension of the approach used by the Simulation Model Development Environment (SMDE) prototype, developed by Balci and Nance (1987), which used a relational database as the central knowledge-base, as opposed to semantic networks.

The five main AI techniques used in semantic modelling are the semantic network graphical knowledge-base, data-driven natural language understanding and processing, procedural attachment, expert systems and shortest path algorithms. Since semantic modelling is an open architecture, more techniques, possibly from other research areas, could be added in future to improve the simulation environment.

This thesis has also presented a working prototype which implements semantic modelling. This has shown the potential of semantic modelling through its implementation. The prototype may provide a stepping stone to a better prototype. These derived systems could

utilise more modern programming techniques and environments, including object oriented programming and windowed environments (as opposed to the present implementation that used Turbo Pascal under DOS).

Section 7.3 : FUTURE RESEARCH

There is considerable potential for further research in this area, as indicated in the analysis of the working prototype (section 6.4). This includes the investigation of the best type of graphical interface for semantic networks. Additionally, the investigation of the practicality of using passive production rules, as opposed to active production rules, could increase the user-friendliness of the interface and improve the inference engine. Additionally, it may be beneficial to investigate whether the inference engine can be enhanced by the analysis of the results of simulation runs. The investigation of both the syntactic structures and the dictionary of the natural language interface may produce a consistent way to define them, rather than ad hoc techniques presently adopted. The investigation of whether it is possible to use the syntactic structures of the natural language interface to 'work backwards' from a semantic network to produce sentences could lead to a translation capability (between any languages which have a pre-defined syntactic structure). There are also likely to be other additional concepts which would increase the user-friendliness of

semantic networks, such as alternative input techniques or alternative types of knowledge-base attachment techniques (in addition to procedural attachment).

These areas are an indication of the many cross-disciplines and domains which the semantic modelling approach utilises. This reinforces the view that this research has added to the level of understanding of the applicability of artificial intelligence to simulation environments. Hopefully, some of the ideas put forward in this thesis will also benefit the wider computer science field.

APPENDICES

APPENDIX 1

NATURAL LANGUAGE SYNTAX DEFINITION

This appendix contains the data definition of the natural language interface in the SASIM prototype. It is contained in the file NLUP.SYN and can be easily modified by most word-processors.

N.B. ^ indicates keywords
 ~ indicates optional
 > is the semantic translation
 / means "or"
 * ; indicate beginning and ending of syntactic structures
 " " indicate words to be treated as a single label

*SEPARATOR :

```
"^," ~SEPARATOR / ^A / ^AND ~SEPARATOR / ^ALSO;
```

*ENTITY P :

```
~^THE ~^A ~obj_number ^ENTITY ~^WHICH ~^IS ~^CALLED obj_name  
> obj_name ^NUMBER obj_number,  
obj_name "^IS A" ^ENTITY /
```

```
~^THE ~^A ~obj_number obj_name ~^WHICH ^IS ~^AN ^ENTITY  
> obj_name ^NUMBER obj_number,  
obj_name "^IS A" ^ENTITY /
```

```
~^THE ~^A ~obj_number obj_name  
> obj_name ^NUMBER obj_number,  
obj_name "^IS A" ^ENTITY /
```

```
~^THE ^HE
```

```
= obj_name HE=LAST_ENT;
```

*ENTITY FOR ACT :

```
~^THE ~^A ~obj_number ^ENTITY ~^WHICH ~^IS ~^CALLED obj_name  
> obj_name "^IS A" ^ENTITY /
```

```
~^THE ~^A ~obj_number obj_name ~^WHICH ^IS ~^AN ^ENTITY  
> obj_name "^IS A" ^ENTITY /
```

```
~^THE ~^A ~obj_number obj_name  
> obj_name "^IS A" ^ENTITY /
```

```
~^THE ^HE
```

```
= obj_name ~obj_number HE=LAST_ENT;
```

***TIME_EXPRESSION :**
^HOUR / ^MINUTE / ^DAY / ^YEAR / ^MONTH / ^WEEK;

***ACTIVITY_BODY :**
~^AN ^ACTIVITY ~^CALLED obj_activity ~^AGAIN
> obj_activity "^IS A" ^ACTIVITY /

obj_activity ~^AGAIN ~^WHICH ~^IS ~^AN ^ACTIVITY
> obj_activity "^IS A" ^ACTIVITY /

obj_activity ~^AGAIN
> obj_activity "^IS A" ^ACTIVITY /
^THIS ^ACTIVITY

= obj_activity THIS=LAST_ACT;

***STRING :**
obj_word ~STRING

= obj_word STRING;

***MULTI_ENTITY_ACT :**
ENTITY_FOR_ACT /

ENTITY_FOR_ACT SEPARATOR ~MULTI_ENTITY_ACT

= ENTITY_FOR_ACT MULTI_ENTITY_ACT;

***ONE_INSTANCE_ACT :**
ENTITY_FOR_ACT /

ENTITY_FOR_ACT ^AS ^A rel_name

= ENTITY_FOR_ACT ~rel_name;

***INSTANCE_ACT :**
ONE_INSTANCE_ACT /

ONE_INSTANCE_ACT SEPARATOR INSTANCE_ACT

= ONE_INSTANCE_ACT INSTANCE_ACT;

***MULTI_ENTITY :**

ENTITY_P /

ENTITY_P SEPARATOR ~MULTI_ENTITY

= ENTITY_P MULTI_ENTITY;

***ONE_INSTANCE :**

ENTITY_P /

ENTITY_P ^AS ^A rel_name

= ENTITY_P ~rel_name;

***INSTANCE :**

ONE_INSTANCE /

ONE_INSTANCE SEPARATOR INSTANCE

= ONE_INSTANCE INSTANCE;

***ACTIVITY_TIME :**

^EVERY obj_number TIME_EXPRESSION /

^TAKES obj_number TIME_EXPRESSION

= obj_number ~EVERY;

***ACTIVITY_DESC :**

ACTIVITY_BODY ~ACTIVITY_TIME

> ACTIVITY_BODY ^DURATION ACTIVITY_TIME /

ACTIVITY_BODY ACTIVITY_TIME ~SEPARATOR ^USES INSTANCE_ACT

> ACTIVITY_BODY ^DURATION ACTIVITY_TIME,
INSTANCE(ACTIVITY_BODY, INSTANCE_ACT) DUMMY DUMMY /

ACTIVITY_BODY ~SEPARATOR ~^USES INSTANCE_ACT ~SEPARATOR
~ACTIVITY_TIME

> ACTIVITY_BODY ^DURATION ACTIVITY_TIME,
INSTANCE(ACTIVITY_BODY, INSTANCE_ACT) DUMMY DUMMY

= ACTIVITY_BODY;

```

*ACTIVITY_P :
  ACTIVITY_DESC /

  ACTIVITY_DESC SEPARATOR ACTIVITY_P

  = ACTIVITY_DESC ACTIVITY_P;

*SENTENCE :
  ACTIVITY_P ~^HAS ^PRIORITY ~^OVER ~^ON ACTIVITY_P_2 "^."
    > ACTIVITY_P ^PRIORITY ACTIVITY_P_2 /

  ^THERE ^IS ACTIVITY_P "^." /

  ^THERE ^IS MULTI_ENTITY "^." /

  ACTIVITY_P "^." /

  obj_word rel_name obj_word2 "^."
    > obj_word rel_name obj_word2 /

  ENTITY_P ~^IS ACTIVITY_P "^."
    > ACTIVITY_P ^NEEDS INSTANCE(ACTIVITY_P,ENTITY_P) /

  ENTITY_P ~^IS ^THEN ~^PRECEDES ~^TO ACTIVITY_P "^."
    > INSTANCE(LAST ACT,ENTITY_P) ^PRECEDES
      INSTANCE(ACTIVITY_P,ENTITY_P) /

  ENTITY_P ~^IS ^THEN ~^PRECEDES ~^TO ACTIVITY_P ^IF STRING "^."
    > INSTANCE(LAST ACT,ENTITY_P) ^PRECEDES "IF &STRING",
      "IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P,ENTITY_P),
      "IF &STRING" "^IS A" ^DECISION /

  ENTITY_P ~^IS ^THEN ~^PRECEDES ~^TO ACTIVITY_P ^IF STRING ^ELSE
  ~^HE ACTIVITY_P_2 "^."
    > INSTANCE(LAST ACT,ENTITY_P) ^PRECEDES
      INSTANCE(ACTIVITY_P_2,ENTITY_P),
      INSTANCE(LAST ACT,ENTITY_P) ^PRECEDES "IF &STRING",
      "IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P,ENTITY_P),
      "IF &STRING" "^IS A" ^DECISION /

  ENTITY_P ^AFTER ~^HE ~^IS ~^BEEN ACTIVITY_P ~SEPARATOR ~^IS ~^HE
  ~^CAN ~^THEN ~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 "^."
    > INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES
      INSTANCE(ACTIVITY_P_2,ENTITY_P) /

```

```
ENTITY_P ^AFTER ~^HE ~^IS ~^BEEN ACTIVITY_P ~SEPARATOR ~^IS ~^HE
~^CAN ~^THEN ~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 ^IF
STRING "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES "IF &STRING",
"IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P_2,ENTITY_P),
"IF &STRING" "^IS A" ^DECISION /
```

```
ENTITY_P ^AFTER ~^HE ~^IS ~^BEEN ACTIVITY_P ~SEPARATOR ~^IS ~^HE
~^CAN ~^THEN ~^PROCEED ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 ^IF
STRING ^ELSE ~^HE ACTIVITY_P_3 "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES
INSTANCE(ACTIVITY_P_3,ENTITY_P),
INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES "IF &STRING",
"IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P_2,ENTITY_P),
"IF &STRING" "^IS A" ^DECISION /
```

```
^AFTER ENTITY_P ACTIVITY_P ~SEPARATOR ~^HE ~^CAN ~^THEN
~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES
INSTANCE(ACTIVITY_P_2,ENTITY_P) /
```

```
^AFTER ENTITY_P ACTIVITY_P ~SEPARATOR ~^HE ~^CAN ~^THEN
ACTIVITY_P_2
~^IF STRING "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES "IF &STRING",
"IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P_2,ENTITY_P),
"IF &STRING" "^IS A" ^DECISION /
```

```
^AFTER ENTITY_P ACTIVITY_P ~SEPARATOR ~^HE ~^CAN ~^THEN
~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 ^IF STRING ^ELSE
~^HE ACTIVITY_P_3 "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES
INSTANCE(ACTIVITY_P_3,ENTITY_P),
INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES "IF &STRING",
"IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P_2,ENTITY_P),
"IF &STRING" "^IS A" ^DECISION /
```

```
^AFTER ACTIVITY_P ~SEPARATOR ENTITY_P ~^HE ~^CAN ~^THEN
~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES
INSTANCE(ACTIVITY_P_2,ENTITY_P) /
```

```
^AFTER ACTIVITY_P ~SEPARATOR ENTITY_P ~^HE ~^CAN ~^THEN
ACTIVITY_P_2 ^IF STRING "^."
```

```
> INSTANCE(ACTIVITY_P,ENTITY_P) ^PRECEDES "IF &STRING",
"IF &STRING" ^PRECEDES INSTANCE(ACTIVITY_P_2,ENTITY_P),
"IF &STRING" "^IS A" ^DECISION /
```

```
^AFTER ACTIVITY_P ~SEPARATOR ENTITY_P ~^HE ~^CAN ~^THEN
~^PRECEDES ~^TO ~^BE ~^USED ~^IN ACTIVITY_P_2 ^IF STRING ^ELSE
~^HE ACTIVITY_P_3 "^. "
  > INSTANCE( ACTIVITY_P, ENTITY_P) ^PRECEDES
INSTANCE( ACTIVITY_P_3, ENTITY_P),
  INSTANCE( ACTIVITY_P, ENTITY_P) ^PRECEDES "IF &STRING",
  "IF &STRING" ^PRECEDES INSTANCE( ACTIVITY_P_2, ENTITY_P),
  "IF &STRING" "^IS A" ^DECISION.
```

APPENDIX 2

NATURAL LANGUAGE CONVERSION MASKS

NOTE : First word on every line contains the mask which is compared to the word being tested. A '*' means any number of characters. If a match is found, the entered word is replaced by the second word on the line (if no second word is present then the original word would be ignored). The list is held in file CONVERT.DAT.

A	
AGAIN	
ALSO	
AN	
AND	
ONE	1
ARE	
AT	
AVAILABLE	
BE	
BEING	
BEGINS	
BETWEEN	
BOTH	
CALLED	
CAN	
CURRENTLY	
EIGHT	8
FOUR	4
FIVE	5
FOR	DURATION
FURTHER	
HAPPENS	
HAS	
HAVE	
HUNDRED	100
IN	
INSIDE	
INTO	
IS	
IT	THEY
LAST	
OCCURS	
OF	
ON	USES
OVER	
PRECEDES	PRECEDES
*MEN	MAN
MINUTE	
MINUTES	

MUST	
NEEDED	
NINE	9
NOW	
OTHER	
SEVEN	7
SIX	6
TEN	10
THE	
THERE	
THEREARE	
THIRTY	30
THOUSAND	1000
THREE	3
TIMES	TIMES
TO	
TWENTY	20
TWO	2
USED	
WILL	
WHICH	
WHO	
*ERVING	ERVICE
*IVAL	IVE
*NING	NE
*PPIN	GP
*TTIN	GT
*VING	VE
USING	USES
*ING	
*IES	Y
*SS	SS
*SSES	SS
USES	USES
*ES	E
*IS	IS
*CEED	CEED
*EED	EED
*ERVED	ERVICE
*IED	Y
*CED	CE
*LLED	LL
*LED	LE
*PPED	P
*VED	VE
*NED	NE
*ED	
AS	AS
*S	
*UNK	INK
*LY	LE

The following words are synonyms and are translated by SASIM:

BY	USES
FOR	DURATION
LATTER	THEY
LEAVE	EXIT
NEED	USES
OBJECT	ENTITY
PEOPLE	ENTITY
PROCEED	THEN
TAKE	DURATION
THEM	THEY
WITH	USES
WHEN	AFTER

APPENDIX 3

TEST OF SASIM ON THE PUB

APPENDIX 3.1 : NLUP TRANSLATION

To illustrate this, I will try to formulate a simple PUB example as an english representation acceptable to SASIM, then as represented in a semantic network (which was produced by SASIM):

INPUT ENGLISH REPRESENTATION :

There is an activity called service.
This activity uses one barmaid as a server.
This activity also needs 1 customer, a glass as a cup and takes 5 minutes.
Customers arrive every negexp(10,5) minutes.
They are then served.
Customers after being served, then drink for 10 minutes, with a glass.
The latter are then washed by the barmaids.
Service has priority over washing.
Customers after drinking proceed to rest for three minutes.
They are then served again if they have drunk less than three times, else they exit.
Glasses after washing can then be used in serving.
There are 10 glasses and ten barmaids and one dog.
Washing takes 15 minutes.

SEMANTIC NETWORK TRANSLATION :

The English sentence was :

There is an activity called service.

TRANSLATION IS
=====

SERVICE IS A ACTIVITY

The English sentence was :

This activity uses one barmaid as a server.

TRANSLATION IS
=====

BARMAID (SERVICE) NUMBER 1
SERVICE SERVER BARMAID (SERVICE)
BARMAID (SERVICE) IS A BARMAID
BARMAID IS A ENTITY

The English sentence was :

This activity also needs 1 customer, a glass as a cup and takes 5 minutes.

TRANSLATION IS
=====

SERVICE DURATION 5
SERVICE CUP GLASS (SERVICE)
GLASS (SERVICE) IS A GLASS
GLASS IS A ENTITY
CUSTOMER (SERVICE) NUMBER 1
SERVICE NEEDS CUSTOMER (SERVICE)
CUSTOMER (SERVICE) IS A CUSTOMER
CUSTOMER IS A ENTITY

The English sentence was :

Customers arrive every $\text{negexp}(10,5)$ minutes.

TRANSLATION IS
=====

CUSTOMER DOOR (ARRIVE) NUMBER 1
ARRIVE NEEDS CUSTOMER DOOR (ARRIVE)
CUSTOMER DOOR (ARRIVE) IS A CUSTOMER DOOR
CUSTOMER DOOR IS A ENTITY
ARRIVE DURATION NEGEXP(10,5)
ARRIVE NEEDS CUSTOMER (ARRIVE)
CUSTOMER (ARRIVE) IS A CUSTOMER
ARRIVE IS A ACTIVITY

The English sentence was :

They are then served.

TRANSLATION IS

=====

CUSTOMER (ARRIVE) PRECEDES CUSTOMER (SERVICE)

The English sentence was :

Customers after being served, then drink for 10 minutes,
with a glass.

TRANSLATION IS

=====

DRINK NEEDS GLASS (DRINK)
GLASS (DRINK) IS A GLASS
DRINK DURATION 10
CUSTOMER (SERVICE) PRECEDES CUSTOMER (DRINK)
DRINK NEEDS CUSTOMER (DRINK)
CUSTOMER (DRINK) IS A CUSTOMER
DRINK IS A ACTIVITY

The English sentence was :

The latter are then washed by the barmaids.

TRANSLATION IS

=====

WASH NEEDS BARMAID (WASH)
BARMAID (WASH) IS A BARMAID
GLASS (DRINK) PRECEDES GLASS (WASH)
WASH NEEDS GLASS (WASH)
GLASS (WASH) IS A GLASS
WASH IS A ACTIVITY

The English sentence was :

Service has priority over washing.

TRANSLATION IS

=====

SERVICE PRIORITY WASH

The English sentence was :

Customers after drinking proceed to rest for three minutes.

TRANSLATION IS

=====

REST DURATION 3
CUSTOMER (DRINK) PRECEDES CUSTOMER (REST)
REST NEEDS CUSTOMER (REST)
CUSTOMER (REST) IS A CUSTOMER
REST IS A ACTIVITY

The English sentence was :

They are then served again if they have drunk less than three times, else they exit.

TRANSLATION IS

=====

CUSTOMER (REST) PRECEDES CUSTOMER (EXIT)
EXIT NEEDS CUSTOMER (EXIT)
CUSTOMER (EXIT) IS A CUSTOMER
EXIT IS A ACTIVITY
IF THEY DRUNK LESS THAN 3 TIMES IS A DECISION
IF THEY DRUNK LESS THAN 3 TIMES PRECEDES CUSTOMER (SERVICE)
CUSTOMER (REST) PRECEDES IF THEY DRUNK LESS THAN 3 TIMES

The English sentence was :

Glasses after washing can then be used in serving.

TRANSLATION IS

=====

GLASS (WASH) PRECEDES GLASS (SERVICE)

The English sentence was :

There are 10 glasses and ten barmaids and one dog.

TRANSLATION IS
=====

DOG IS A ENTITY
DOG NUMBER 1
BARMAID NUMBER 10
GLASS NUMBER 10

The English sentence was :

Washing takes 15 minutes.

TRANSLATION IS
=====

WASH DURATION 15

APPENDIX 3.2 : LIST OF LINKS

This section presents a list of all links created by the automatic NLUP translation of the pub problem, shown in appendix 3.1.

WASH DURATION 15
DOG IS A ENTITY
DOG NUMBER 1
BARMAID NUMBER 10
GLASS NUMBER 10
GLASS (WASH) PRECEDES GLASS (SERVICE)
CUSTOMER (REST) PRECEDES CUSTOMER (EXIT)
EXIT NEEDS CUSTOMER (EXIT)
CUSTOMER (EXIT) IS A CUSTOMER
EXIT IS A ACTIVITY
IF THEY DRUNK LESS THAN 3 TIMES IS A DECISION
IF THEY DRUNK LESS THAN 3 TIMES PRECEDES CUSTOMER (DRINK)
CUSTOMER (REST) PRECEDES IF THEY DRUNK LESS THAN 3 TIMES
REST DURATION 3
CUSTOMER (DRINK) PRECEDES CUSTOMER (REST)
REST NEEDS CUSTOMER (REST)
CUSTOMER (REST) IS A CUSTOMER
REST IS A ACTIVITY
SERVICE PRIORITY WASH
WASH NEEDS BARMAID (WASH)
BARMAID (WASH) IS A BARMAID
GLASS (DRINK) PRECEDES GLASS (WASH)
WASH NEEDS GLASS (WASH)
GLASS (WASH) IS A GLASS
WASH IS A ACTIVITY
DRINK NEEDS GLASS (DRINK)
GLASS (DRINK) IS A GLASS
DRINK DURATION 10
CUSTOMER (SERVICE) PRECEDES CUSTOMER (DRINK)
DRINK NEEDS CUSTOMER (DRINK)
CUSTOMER (DRINK) IS A CUSTOMER
DRINK IS A ACTIVITY
CUSTOMER (ARRIVE) PRECEDES CUSTOMER (SERVICE)
CUSTOMER DOOR (ARRIVE) NUMBER 1
ARRIVE NEEDS CUSTOMER DOOR (ARRIVE)
CUSTOMER DOOR (ARRIVE) IS A CUSTOMER_DOOR
CUSTOMER_DOOR IS A ENTITY
ARRIVE DURATION NEGEXP(10,5)
ARRIVE NEEDS CUSTOMER (ARRIVE)
CUSTOMER (ARRIVE) IS A CUSTOMER
ARRIVE IS A ACTIVITY
SERVICE DURATION 5
SERVICE CUP GLASS (SERVICE)
GLASS (SERVICE) IS A GLASS
GLASS IS A ENTITY

CUSTOMER (SERVICE) NUMBER 1
SERVICE NEEDS CUSTOMER (SERVICE)
CUSTOMER (SERVICE) IS A CUSTOMER
CUSTOMER IS A ENTITY
BARMAID (SERVICE) NUMBER 1
SERVICE SERVER BARMAID (SERVICE)
BARMAID (SERVICE) IS A BARMAID
BARMAID IS A ENTITY
SERVICE IS A ACTIVITY

APPENDIX 3.3 : LIST OF OBJECTS

This section presents a list of all objects created by the automatic NLUP translation of the pub problem, shown in appendix 3.1.

1
10
3
5
ACTIVITY
ARRIVE
BARMAID
BARMAID (SERVICE)
BARMAID (WASH)
CUSTOMER
CUSTOMER (ARRIVE)
CUSTOMER (DRINK)
CUSTOMER (EXIT)
CUSTOMER (REST)
CUSTOMER (SERVICE)
CUSTOMER DOOR
CUSTOMER DOOR (ARRIVE)
DECISION
DOG
DRINK
ENTITY
EXIT
GLASS
GLASS (DRINK)
GLASS (SERVICE)
GLASS (WASH)
IF THEY DRUNK LESS THAN 3 TIMES
NEGEXP(10,5)
REST
SERVICE
WASH

APPENDIX 3.4 : LIST OF RELATIONS

This section presents a list of all relationships created by the automatic NLUP translation of the pub problem, shown in appendix 3.1.

NAME	LEVEL
CUP	0
DURATION	0
IS A	2
NEEDS	0
NUMBER	3
PRECEDES	1
PRIORITY	0
SERVER	0

APPENDIX 3.5 : OUTPUT FROM VALIDATE OPTION

This section presents the result of the validate option run on the knowledge-base created by the automatic NLUP translation of the pub problem, shown in appendix 3.1.

3.5.1 : PROBABILITY SECTION

ENTER PROBABILITY OF IF THEY DRUNK LESS THAN 3 TIMES (%) : 66

3.5.2 : ENTITY CYCLE SUMMARY

ACTIVITIES OF ***FACILITY *** BARMAID
=====

SERVICE
HAS PRIORITY OVER ==>
WASH

ACTIVITIES OF CUSTOMER
=====

CUSTOMER (EXIT) PRECEDES CUSTOMER (ARRIVE)

THE ABOVE LINK HAS BEEN ADDED

ARRIVE
FOLLOWED BY ==>
SERVICE
FOLLOWED BY ==>
DRINK
FOLLOWED BY ==>
REST
FOLLOWED BY ==>
EXIT

ENTER NUMBER OF CUSTOMER IN THE SYSTEM : 30

THERE ARE 30 CUSTOMER TO ALLOCATE
ENTER NUMBER AT CUSTOMER (ARRIVE) : 30

ACTIVITIES OF ***FACILITY *** CUSTOMER_DOOR
=====

ARRIVE

ENTER NUMBER OF CUSTOMER_DOOR IN THE SYSTEM : 1

ACTIVITIES OF DOG
=====

COULD NOT COMPLETE CYCLE FOR DOG
PRESS <ESC> TO CONTINUE

**** NONE ****

ACTIVITIES OF GLASS
=====

GLASS (SERVICE) PRECEDES GLASS (DRINK)

THE ABOVE LINK HAS BEEN ADDED

DRINK
FOLLOWED BY ==>
WASH
FOLLOWED BY ==>
SERVICE

THERE ARE 10 GLASS TO ALLOCATE
ENTER NUMBER AT GLASS (DRINK) :

THERE ARE 10 GLASS TO ALLOCATE
ENTER NUMBER AT GLASS (WASH) :

THERE ARE 10 GLASS TO ALLOCATE
ENTER NUMBER AT GLASS (SERVICE) : 10

3.5.3 : ACTIVITY SUMMARY

ENTITIES INVOLVED IN ARRIVE

=====

CUSTOMER DOOR
CUSTOMER

ENTITIES INVOLVED IN DRINK

=====

GLASS
CUSTOMER

ENTITIES INVOLVED IN EXIT

=====

ENTER DURATION OF EXIT 0

CUSTOMER

ENTITIES INVOLVED IN REST

=====

CUSTOMER

ENTITIES INVOLVED IN SERVICE

=====

GLASS
CUSTOMER
BARMAID

ENTITIES INVOLVED IN WASH

=====

GLASS
BARMAID

DOG IS NOT CONNECTED TO GLASS
PRESS <ESC> TO CONTINUE

APPENDIX 3.6 : PASCAL MODIFICATION TO THE PUB EXAMPLE

An example of the modification required in the pub example to allow customers between 2-4 drinks follows :

```

=====
{$I USERCODE.IN1}
  ATT = RECORD
                                INTERNAL_USE : OCC;
                                NO_DRINKS : INTEGER;
  END;
{$I USERCODE.IN2}
VAR
  DOOR_ARRIVE, DOOR_ARRIVE_ARR,
  CUSTOMER_ARR, ARRIVE, CUSTOMER_REST, CUSTOMER_SERVICE,
  CUSTOMER_EXIT, GLASSES, CUSTOMERS, BARMAIDS,
  GLASS_WASH : OOBJECT;
  DRINK_HIST, SERVE_TS, BARMAID_TS : HHISTOGRAM;

PROCEDURE SET_HOOKS;
BEGIN
  DOOR_ARRIVE := LOCATE_OBJECT('DOOR_ARRIVE');
  DOOR_ARRIVE_ARR := LOCATE_OBJECT('DOOR_ARRIVE (ARRIVE)');
  CUSTOMER_ARR := LOCATE_OBJECT('CUSTOMER (ARRIVE)');
  CUSTOMER_REST := LOCATE_OBJECT('CUSTOMER (REST)');
  CUSTOMER_SERVICE := LOCATE_OBJECT('CUSTOMER (SERVICE)');
  CUSTOMER_EXIT := LOCATE_OBJECT('CUSTOMER (EXIT)');
  CUSTOMERS := LOCATE_OBJECT('CUSTOMER');
  GLASSES := LOCATE_OBJECT('GLASS');
  GLASS_WASH := LOCATE_OBJECT('GLASS (WASH)');
  BARMAIDS := LOCATE_OBJECT('BARMAID');
  ARRIVE := LOCATE_OBJECT('ARRIVE');
  IF NOT ERR THEN
    ARRIVE^.CONTROL := USER;
  ICON_MATRIX(0,0,238,221,154,WHITE,BLACK,GLASSES); (*ε 238,221
                                                    Ü 154*)
  ICON_MATRIX(0,0,12,11,1,YELLOW,BLACK,CUSTOMERS); (* Face *)
  DEFINE_CONVEYOR(-1,0,1,0, 2, GREEN, GLASS_WASH); (* X,Y, XSTEP,
                                                    YSTEP, LENGTH, COLOUR, INSTANCE *)
  INITIALISE_HIST(DRINK_HIST,CUSTOMER_ARR,'NUMBER OF DRINKS');
  INITIALISE_TSERIES(SERVE_TS,CUSTOMER_SERVICE,
                    'CUSTSERV',3,0,15);
  INITIALISE_TSERIES(BARMAID_TS,BARMAIDS,'BARMAID',3,0,15);
END;

```

```

FUNCTION GRAPHIC_OBJECT(OBJ : OOBJECT; XG1,YG1 : INTEGER;
                        DRAW : BOOLEAN) : INTEGER;
VAR
  T1,T2 : INTEGER;
  PENTAGON : ARRAY [1..4] OF POINTTYPE;
BEGIN
  IF OBJ^.NAME = 'CUSTOMER (DRINK)' THEN
  BEGIN (* shape of a man *)
    IF DRAW THEN
    BEGIN
      SETCOLOR(WHITE);
      YG1 := YG1-YDOTS_PER_CHAR;
      XG1 := XG1+XDOTS_PER_CHAR;
      CIRCLE(XG1,YG1,ROUND(YDOTS_PER_CHAR*0.2));
      YG1 := YG1+ROUND(YDOTS_PER_CHAR*0.2);
      LINE(XG1,YG1,XG1,YG1+YDOTS_PER_CHAR);
      YG1 := YG1+YDOTS_PER_CHAR;
      LINE(XG1-ROUND(XDOTS_PER_CHAR*0.5),
           YG1+ROUND(YDOTS_PER_CHAR*0.5),XG1,YG1);
      LINE(XG1+ROUND(XDOTS_PER_CHAR*0.5),
           YG1+ROUND(YDOTS_PER_CHAR*0.5),XG1,YG1);
      YG1 := YG1-ROUND(0.66*YDOTS_PER_CHAR);
      LINE(XG1-ROUND(XDOTS_PER_CHAR*0.5),
           YG1,XG1+ROUND(XDOTS_PER_CHAR*0.5),YG1);
    END;
    GRAPHIC_OBJECT := -3;
  END
  ELSE
  IF OBJ^.NAME = 'CUSTOMER (SERVICE)' THEN
  BEGIN (* money sign to indicate payment when served *)
    IF DRAW THEN
    BEGIN
      SETCOLOR(WHITE);
      RECTANGLE(XG1-13,YG1-4,XG1+21,YG1+12);
      SETUSERCHARSIZE(5,5,5,5);
      SETTEXTSTYLE(SMALLFONT,HORIZDIR,USERCHARSIZE);
      SETCOLOR(GREEN);
      OUTTEXTXY(XG1-10,YG1,'MONEY');
    END;
    GRAPHIC_OBJECT := -5;
  END
  ELSE
  IF (OBJ^.NAME = 'GLASS (SERVICE)') OR
     (OBJ^.NAME = 'GLASS (DRINK)') THEN
  BEGIN (* beer glass shape *)
    IF DRAW THEN
    BEGIN
      SETCOLOR(BROWN);
      setfillstyle(solidfill,BROWN);
      PENTAGON[1].X := XG1+2;
      PENTAGON[1].Y := YG1+ROUND(YDOTS_PER_CHAR*0.5);
    END;
  END;

```



```

        PENTAGON[2].X := XG1+2*XDOTS_PER_CHAR-2;
        PENTAGON[2].Y := YG1+ROUND(YDOTS_PER_CHAR*0.5);
        PENTAGON[3].X := XG1+2*XDOTS_PER_CHAR;
        PENTAGON[3].Y := YG1-YDOTS_PER_CHAR;
        PENTAGON[4].X := XG1;
        PENTAGON[4].Y := YG1-YDOTS_PER_CHAR;
        DRAWPOLY(SIZEOF(PENTAGON) DIV
                SIZEOF(POINTTYPE),PENTAGON);
        FILLPOLY(SIZEOF(PENTAGON) DIV
                SIZEOF(POINTTYPE),PENTAGON);
        PIESLICE(XG1,YG1-YDOTS_PER_CHAR,90,270,2);
        PIESLICE(XG1+2*XDOTS_PER_CHAR,
                YG1-YDOTS_PER_CHAR,0,90,2);
        PIESLICE(XG1+2*XDOTS_PER_CHAR,
                YG1-YDOTS_PER_CHAR,270,360,2);
        SETCOLOR(WHITE);
        setfillstyle(INTERLEAVEFILL,WHITE);
        BAR(XG1,YG1-ROUND(1.2*YDOTS_PER_CHAR),
            XG1+2*XDOTS_PER_CHAR,YG1-YDOTS_PER_CHAR);
    END;
    GRAPHIC_OBJECT := -2;
END
ELSE
    IF OBJ^.FORGROUND_COLOUR = -1 THEN (* default section *)
        GRAPHIC_OBJECT := LENGTH(OBJ^.NAME)
    ELSE
        GRAPHIC_OBJECT := 2;
    END;
END;

PROCEDURE B_EVENT(CURRENT_OBJECT : OOBJECT; OCCUR : OCC);
BEGIN
    IF CURRENT_OBJECT = CUSTOMER_REST THEN
        BEGIN
            DEC(ATTRIBUTE(OCCUR)^.NO_DRINKS);
            IF ATTRIBUTE(OCCUR)^.NO_DRINKS > 0 THEN
                ADD_TO_B_QUEUE(CUSTOMER_SERVICE, OCCUR)
            ELSE
                ADD_TO_B_QUEUE(CUSTOMER_EXIT, OCCUR);
            END
        END
    END;

PROCEDURE C_EVENT(CURRENT_ACTIVITY : OOBJECT);
VAR
    SAMPLE_TIME : INTEGER;
    OCCUR : OCC;
BEGIN
    IF CURRENT_ACTIVITY = ARRIVE THEN
        WHILE (QSIZE(DOOR_ARRIVE) > 0) AND
            (QSIZE(CUSTOMER_ARR) > 0) DO
            BEGIN
                SAMPLE_TIME := SAMPLE_ACT_TIME(ARRIVE);
            END
        END
    END;

```

```

        OCCUR := BEHEAD(CUSTOMER_ARR);
        ATTRIBUTE(OCCUR)^.NO DRINKS := RANDOM(3)+2;
        UPDATE_HIST(DRINK_HIST,ATTRIBUTE(OCCUR)^.NO DRINKS,1);
        ADD_TO_TREE(SAMPLE_TIME,OCCUR,CUSTOMER_ARR);
        ADD_TO_TREE(SAMPLE_TIME,BEHEAD(DOOR_ARRIVE),
            DOOR_ARRIVE_ARR);
    END;
END;

PROCEDURE REPORTS;
BEGIN
    DISPLAY_HISTOGRAM(DRINK_HIST);
    PRINT_HIST(CUSTOMER_SERVICE);
END;

BEGIN
    {BLANK MAIN}
END.

```

APPENDIX 3.7 : SCREEN PRINTOUTS DURING SIMULATION

On starting a simulation run, the system will prompt for the following (assuming they have not been supplied before) :

Question	User Answer
STOP SIMULATION AT	1000
START RECORDING AT	200
PAUSE INTERVAL (AFTER STARTING RECORDING)	200
DO YOU WHICH ONLY A SUMMARY TO BE PRINTED ?	Y

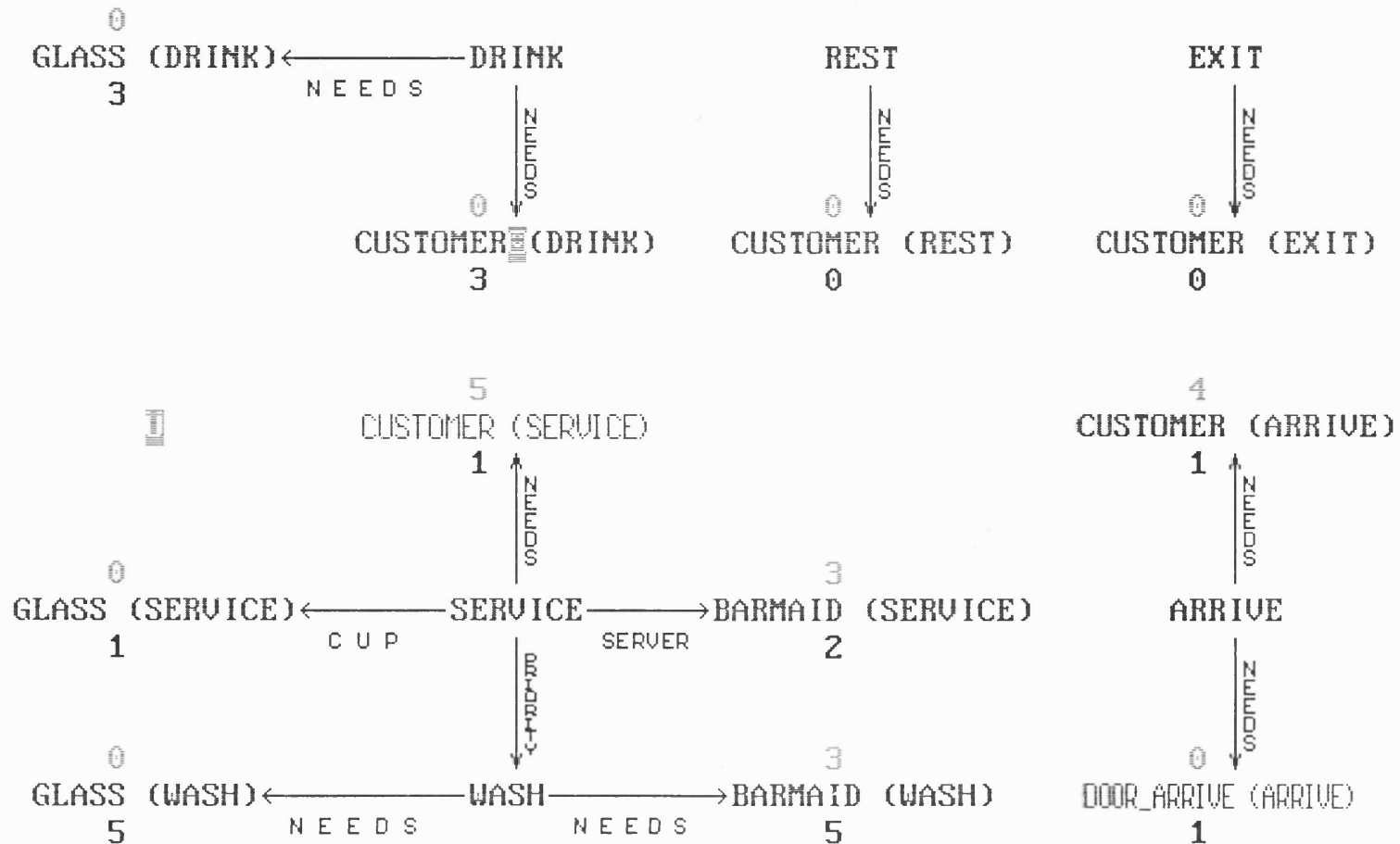
There follows screen printouts of all five levels of the pub produced during a simulation run, together with statistical output.

V1 :

TIME IS 940

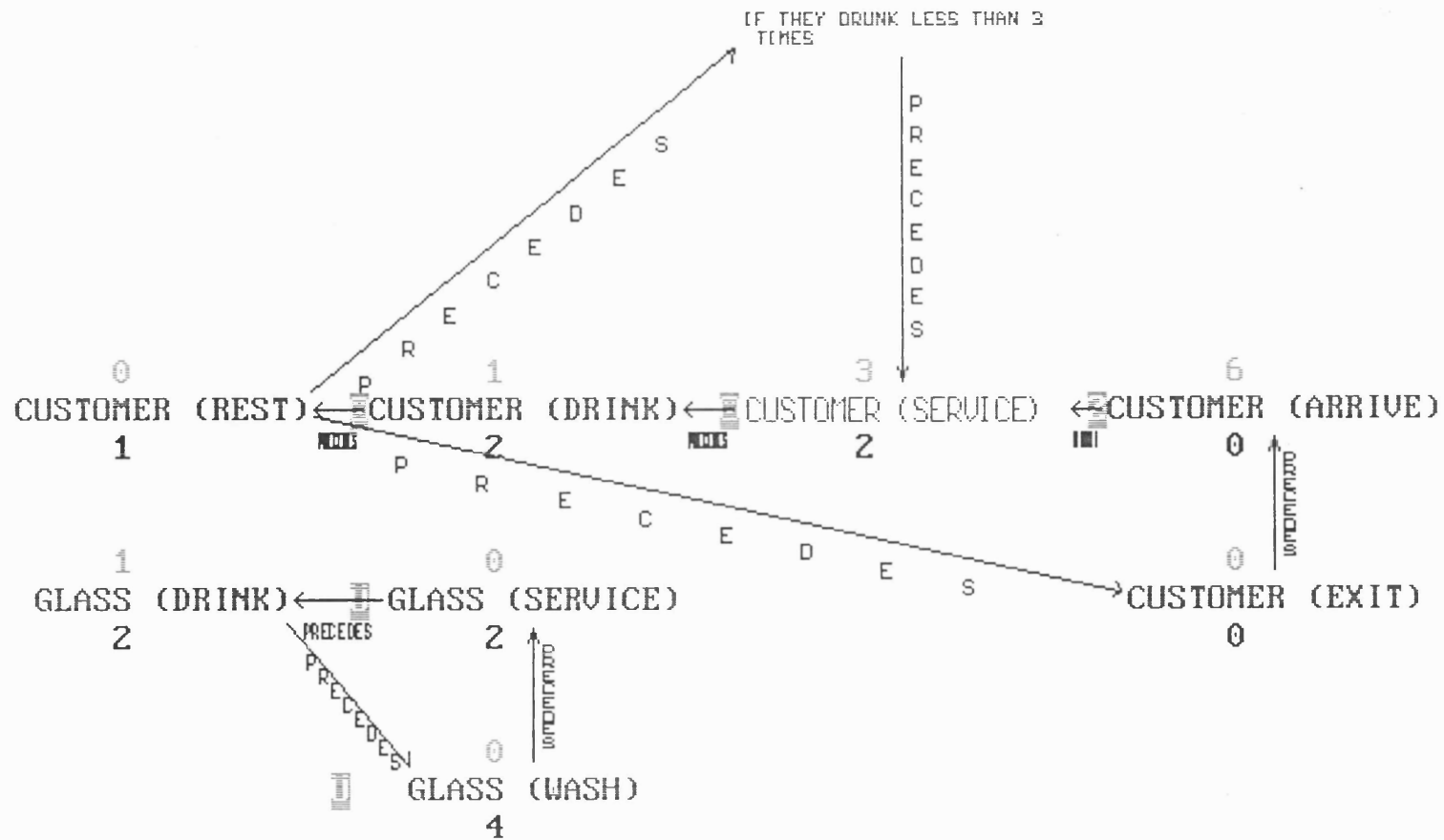
STATUS : RUNNING

SIMULATION DELAY IS 64



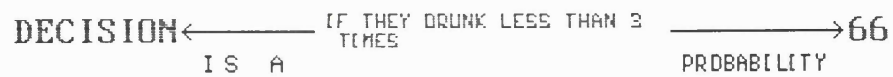
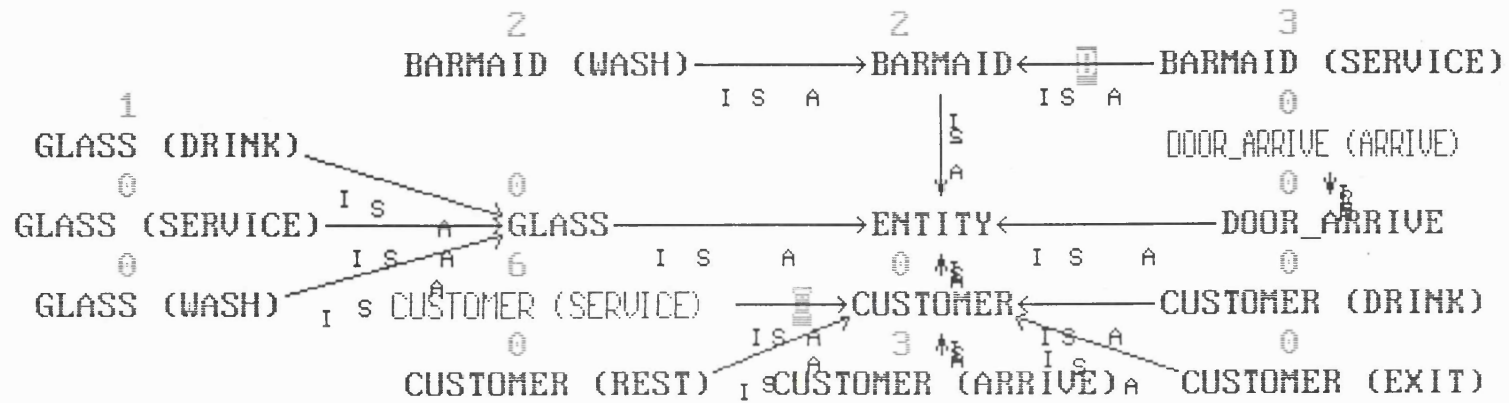
BARMAID (SERVICE) =====> BARMAID

U1 :
 TIME IS 953 STATUS : RUNNING SIMULATION DELAY IS 128



CUSTOMER (SERVICE) =====> CUSTOMER (DRINK)

U1 :
 TIME IS 967 STATUS : RUNNING SIMULATION DELAY IS 128



BARMAID (SERVICE) =====> BARMAID

U1 : REST
TIME IS 977 STATUS : RUNNING SIMULATION DELAY IS 128
NUMBER OF TIMES STARTED 26

REST —————> 3
DURATION

WASH —————> 15
DURATION

SERVICE —————> 5
DURATION

DRINK —————> 10
DURATION

ARRIVE —————> NEGEXP(10,5)
DURATION

4
BARMAID —————> 10
10 NUMBER

0
GLASS —————> 10
10 NUMBER

0
GLASS (SERVICE) —————> 10
0 (INIT NUMBER)

0
CUSTOMER —————> 15
15 NUMBER

2
CUSTOMER (ARRIVE) —————> 15
1 (INIT NUMBER)

BARMAID (WASH) =====> BARMAID

U1 : DRINK

TIME IS 982 STATUS : RUNNING SIMULATION DELAY IS 128


NUMBER OF TIMES STARTED 26

DRINK

CUSTOMERS

SERVICE

0
2



0
2



5
3



0
3



0 0 0

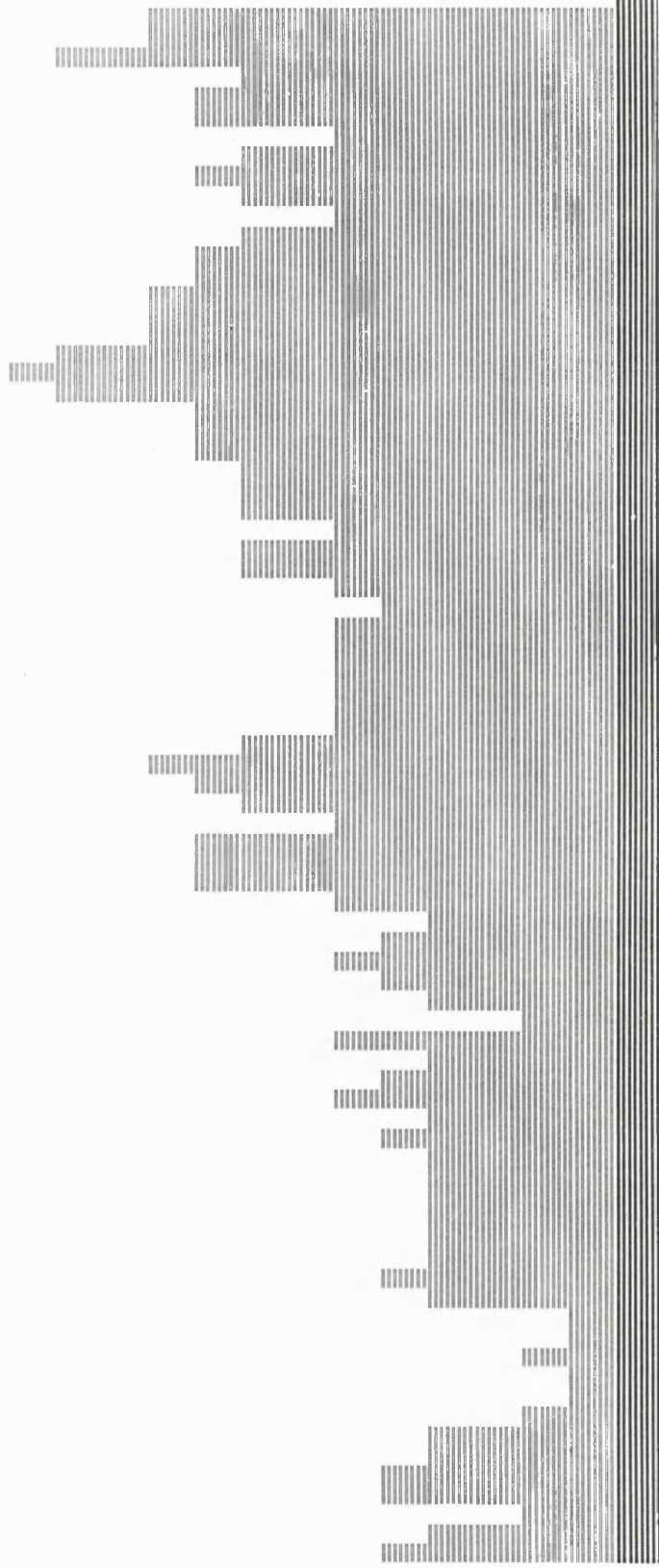


WASHING

BARMAID (SERVICE) =====> BARMAID

1950

THE SERIES OF CIPHERS



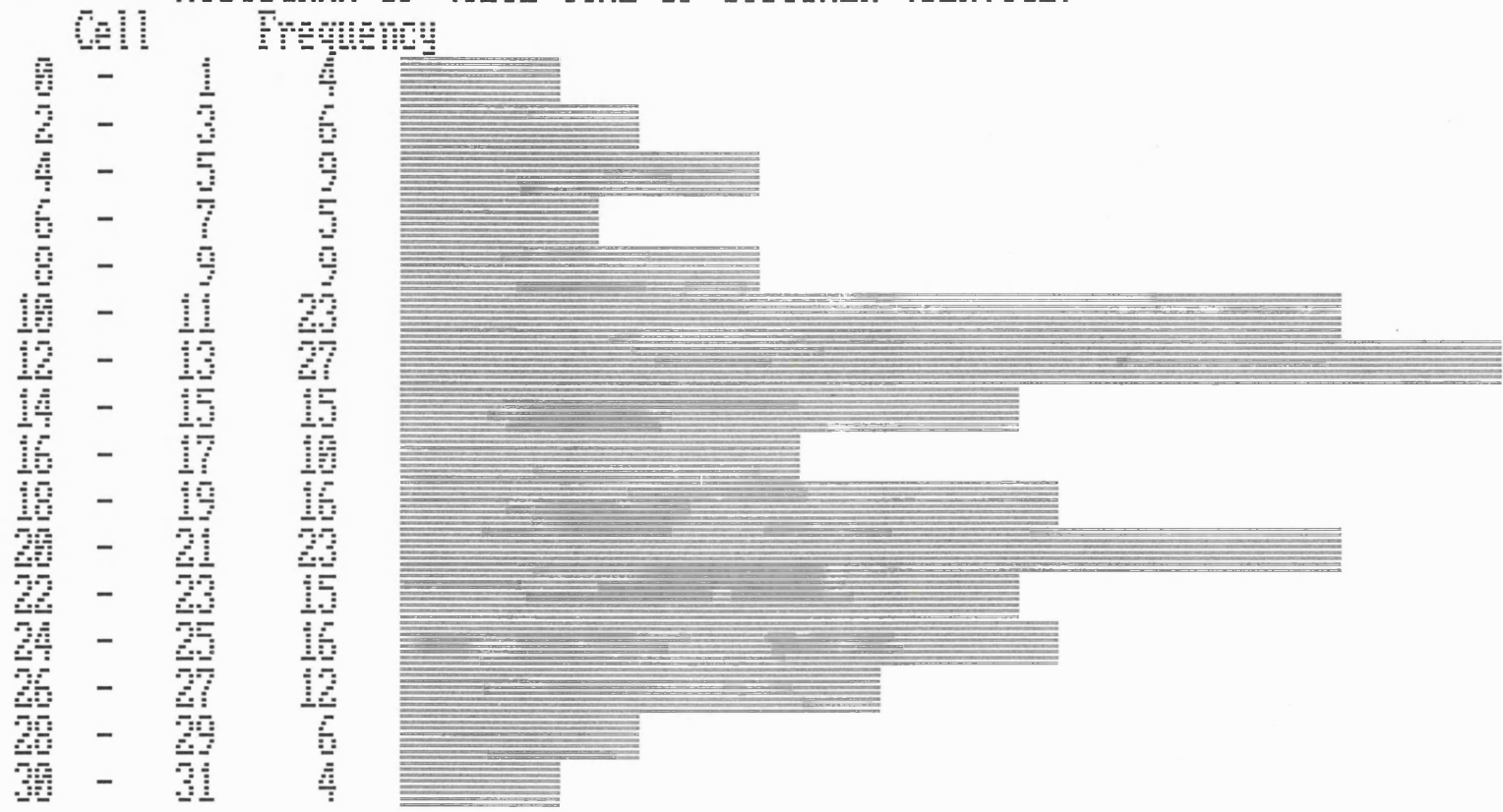
CURRENT TIME 901

----- BASIC STATISTICS OF QUEUE TIME OF CUSTOMER (SERVICE) -----

Mean	=	16.1900
SD	=	7.2893
Variance	=	53.1333
Sum of (freq*obs)	=	3236.0000
Total no. of frequency	=	200
Base of Histogram	=	6
Width of Histogram	=	2

PRESS ANY KEY TO CONTINUE ==>

===== HISTOGRAM OF QUEUE TIME OF CUSTOMER (SERVICE) =====



PRESS ANY KEY TO CONTINUE ==>

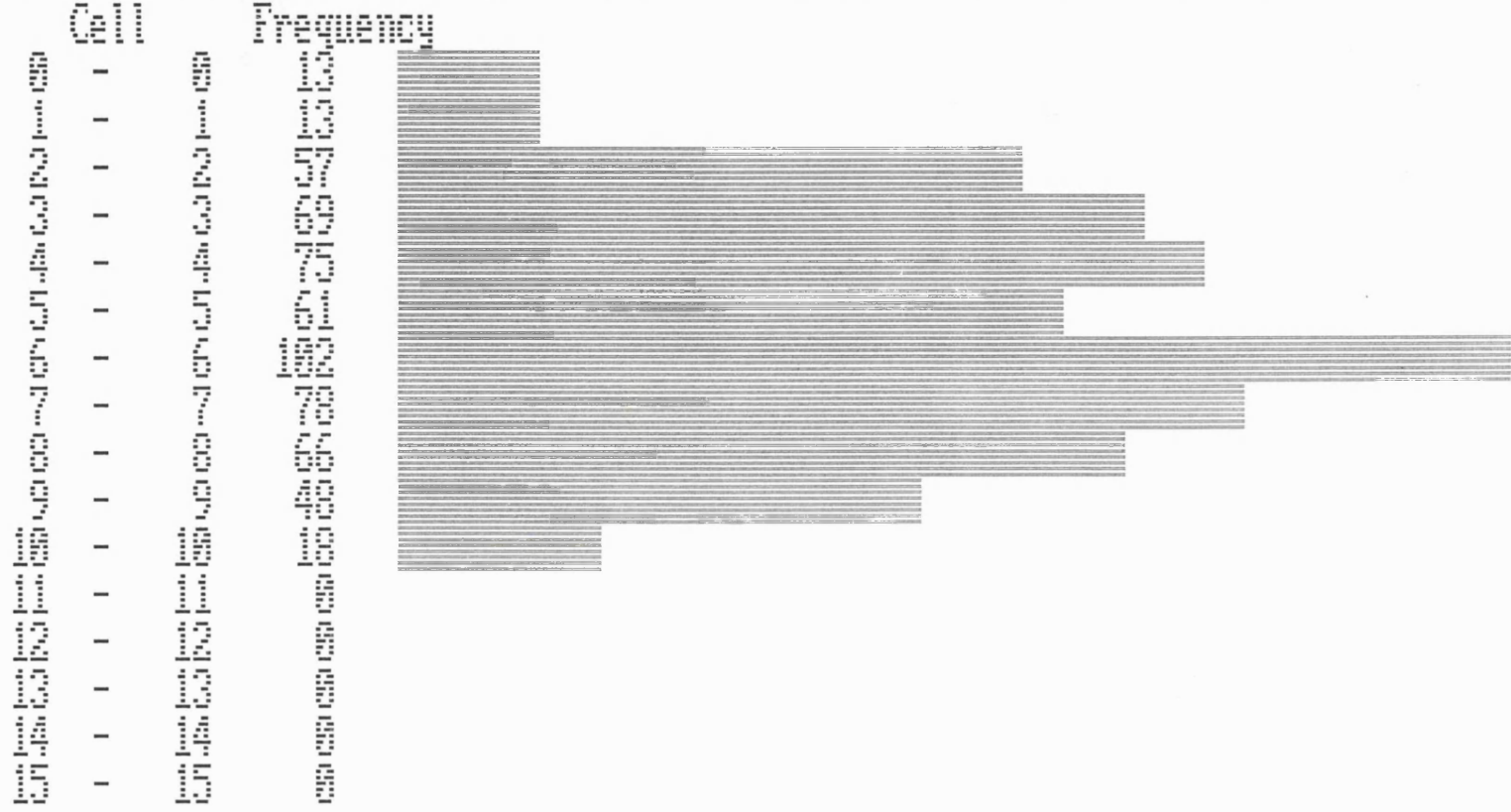
CURRENT TIME 901

----- BASIC STATISTICS OF QUEUE LENGTH OF CUSTOMER (SERVICE) -----

Mean	-	5.3958
SD	-	2.4274
Variance	-	5.8921
Sum of (freq*obs)	-	3237.0000
Total no. of frequency	-	600
Base of Histogram	-	0
Width of Histogram	-	1

PRESS ANY KEY TO CONTINUE ==>

===== HISTOGRAM OF QUEUE LENGTH OF CUSTOMER (SERVICE) =====



PRESS ANY KEY TO CONTINUE ==>

CURRENT TIME 901

----- BASIC STATISTICS OF QUEUE LENGTH OF BARMAID -----

Mean	=	3.3357
SD	=	0.9721
Variance	=	0.9449
Sum of (freq*obs)	=	2002.0000
Total no. of frequency	=	600
Base of Histogram	=	1
Width of Histogram	=	1

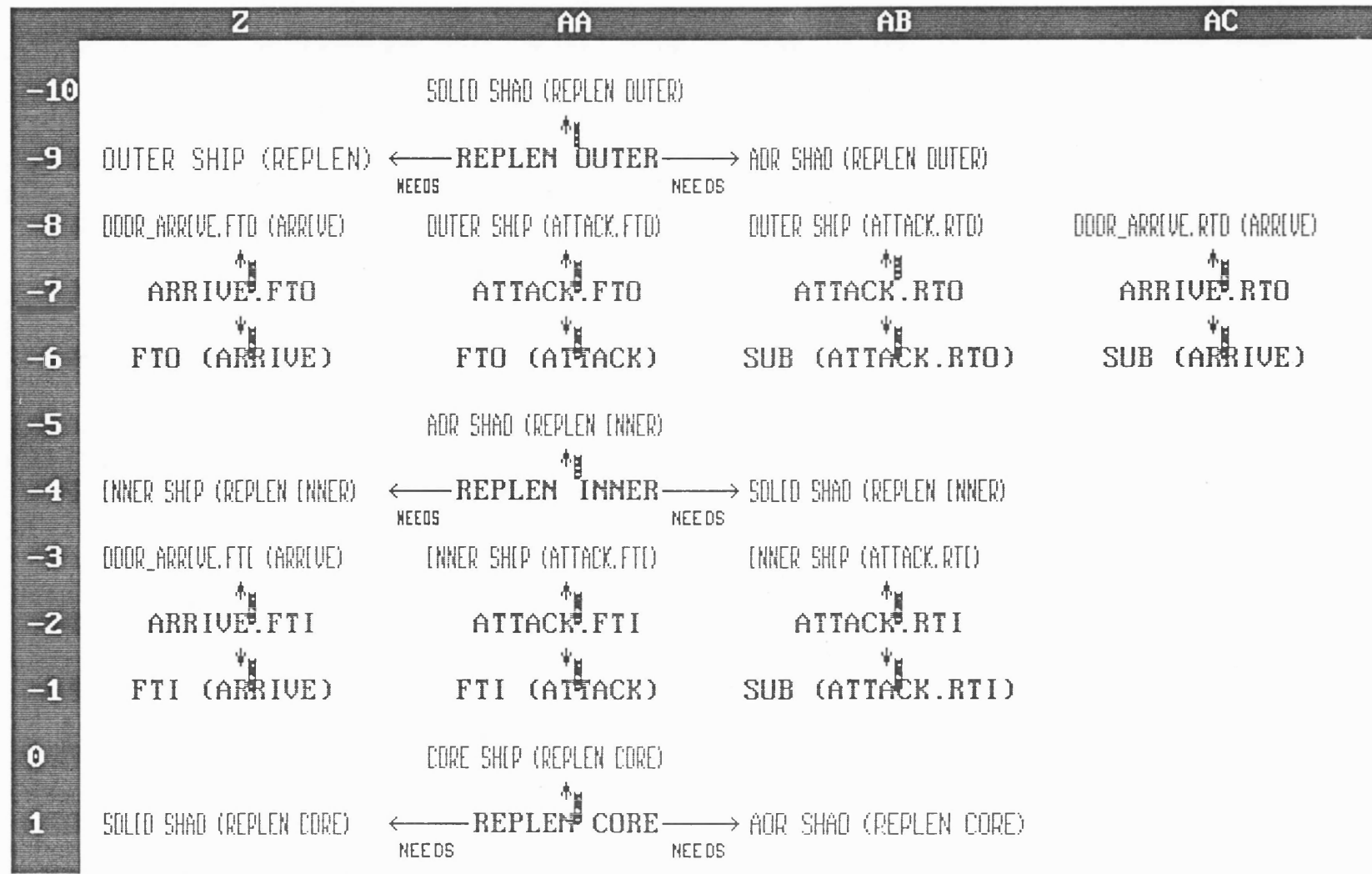
PRESS ANY KEY TO CONTINUE ==>

APPENDIX 4

WAR ARSENAL PROBLEM

The problem "The Effect of Warship and Replenishment Ship Attrition on War Arsenal Requirements" has been described in section 6.3. There follows screen printouts of the semantic network implementation of the problem. However a Pascal modification of the knowledge-base would be required to produce a realistic simulation run.

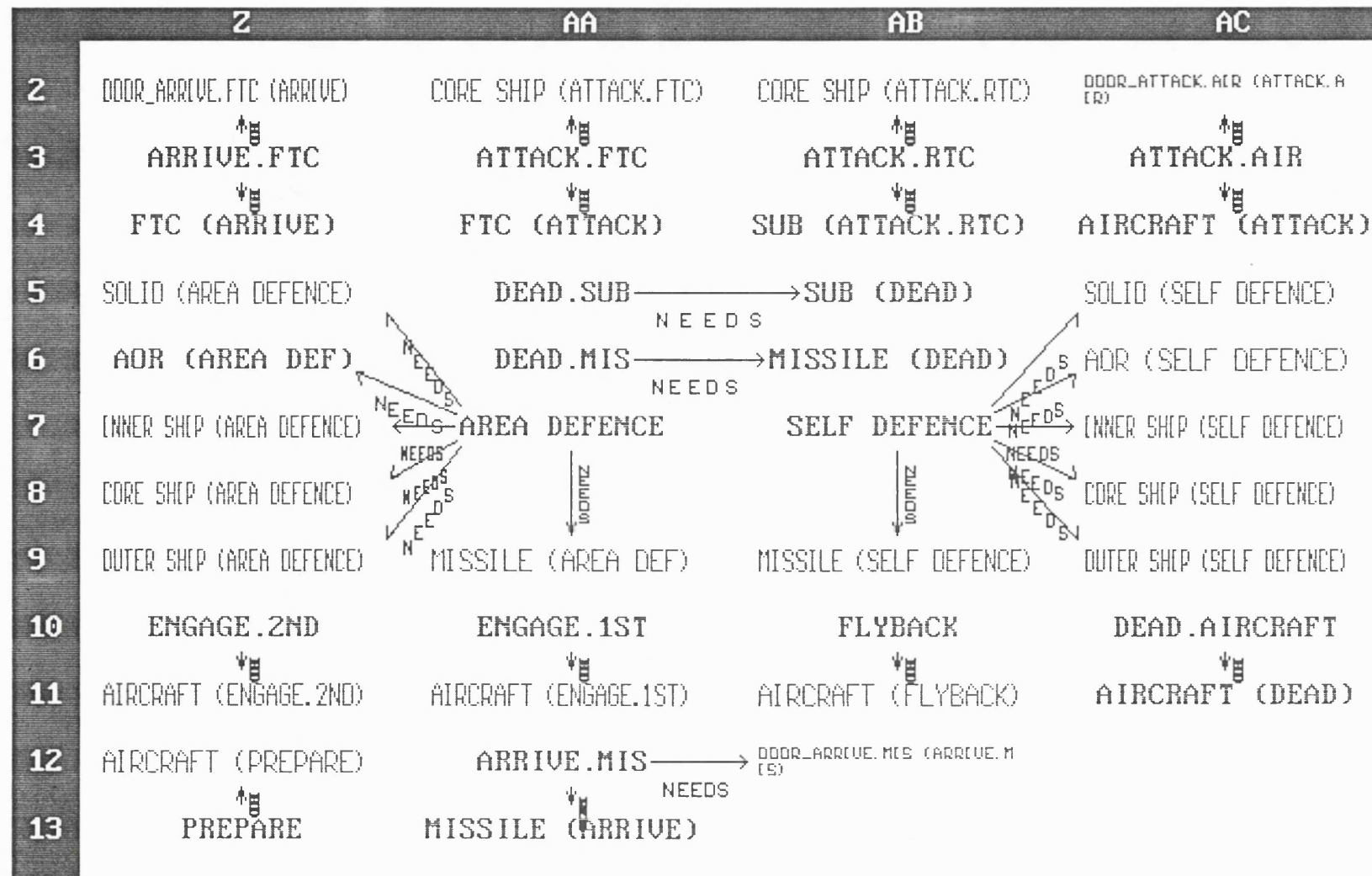
Z-10 :



LEVEL 0 Base Level

Range of this Level : Z-10..AC13 32419

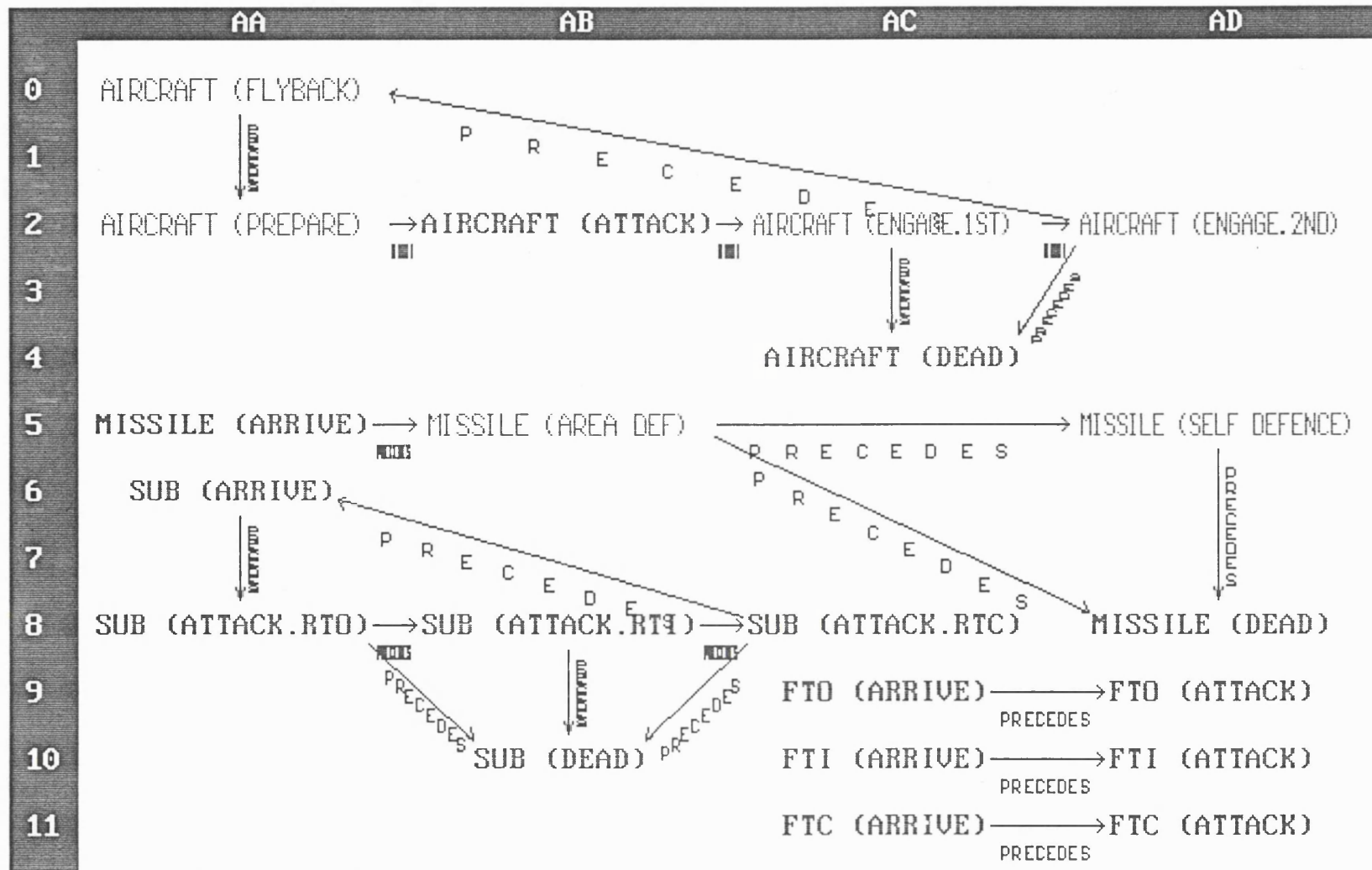
AC13 :



LEVEL 0 Base Level

Range of this Level : Z-10..AC13 32419

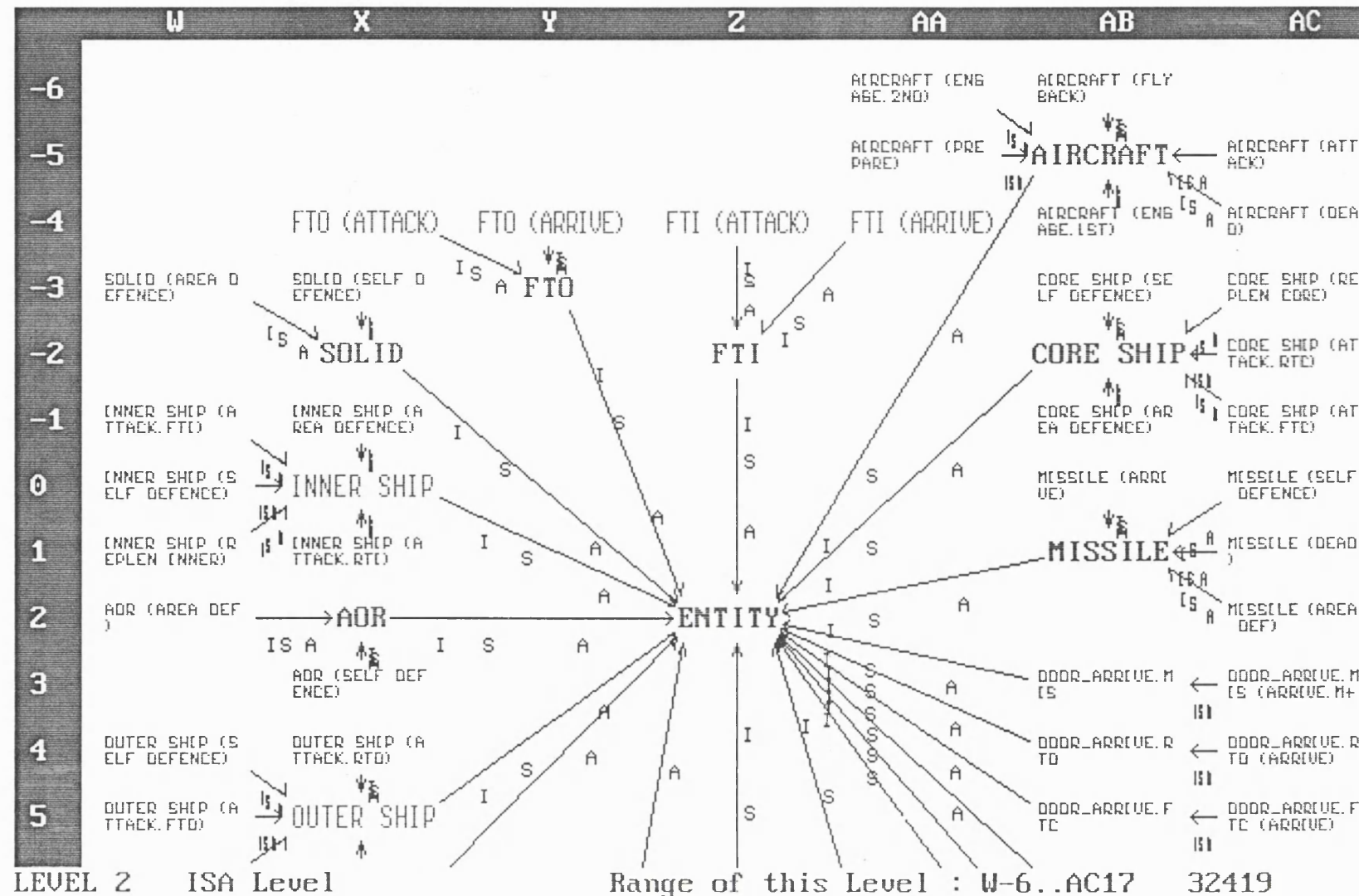
AD11 : FTC (ATTACK)



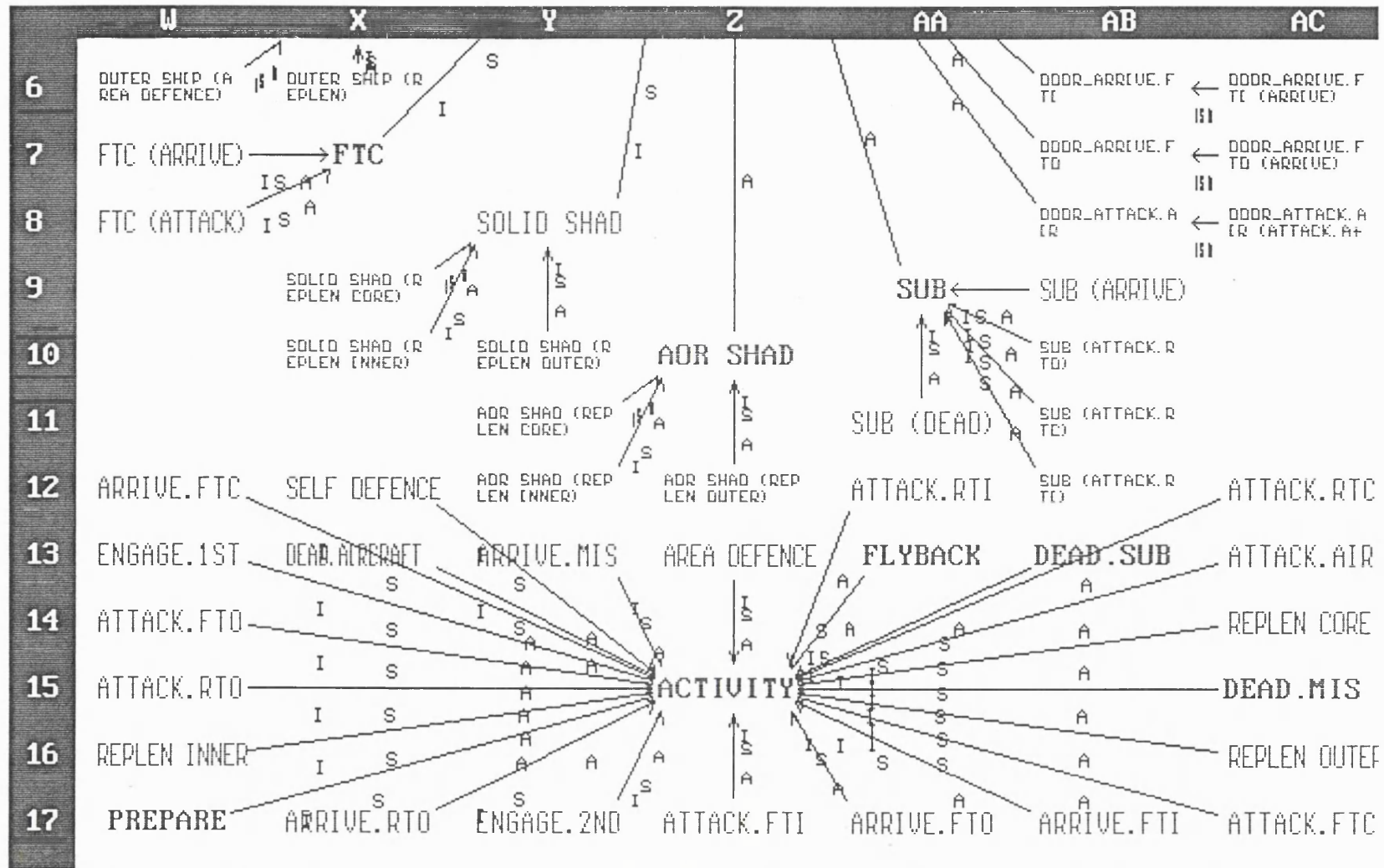
LEVEL 1 Entity Cycle Level

Range of this Level : AA0..AD11 32419

AC5 : DOOR_ARRIVE.FTC (ARRIVE)



AC17 : ATTACK.FTC



LEVEL 2 ISA Level

Range of this Level : W-6..AC17 32419

AD9 :

	AA	AB	AC	AD
0	DOOR_ARRIVE.MIS NUMBER	→1	ARRIVE.RTO DURATION	→10
1				
2	DOOR_ARRIVE.FTO NUMBER	→1	ARRIVE.FTO DURATION	→10
3				
4	DOOR_ARRIVE.FTI NUMBER	→1	ARRIVE.FTI DURATION	→10
5				
6	DOOR_ARRIVE.FTC NUMBER	→1	ARRIVE.FTC DURATION	→10
7				
8	DOOR_ARRIVE.RTO NUMBER	→1	ARRIVE.MIS DURATION	→0
9				
10	DOOR_ATTACK.AIR NUMBER	→1	ATTACK.AIR DURATION	→10
11				

LEVEL 3 Numbers Level

Range of this Level : AA0..AD10 32419

APPENDIX 5

TECHNICAL DATA AND INSTALLATION GUIDE

DATA FILES

*.ENG : Files containing the 'english' text
*.REL : Relationship Data Files

SOURCE UNITS

CYCLEINP.PAS : Contains the code for cycle input.
GLOBAL.PAS : Contains general subroutines.
GLOBAL2.PAS : Contains general subroutines.
GRAPHRTN.PAS : Contains the main graphical routines.
MENU.PAS : Contains the menus.
NLUP.PAS : Contains the NLUP code.
SASIM.PAS : Skeleton file to link the other modules.
SIMULATE.PAS : Contains the main simulation code
USERCODE.PAS : Contains user code to modify the order
of the simulation.
VALIDATE.PAS : The validate section.

OTHER FILES

USERCODE.IN1
USERCODE.IN2 : Small include file
EGAVGA.BGI : Screen driver for an EGA and VGA
compatible screen
LITT.CHR : Fonts file
CONVERT.DAT : Contains conversion masks for SEF

RUNNING

The files containing the executable code needs to be copied onto the same directory, whether on a floppy drive, or in a subdirectory on a hard disk. To start SASIM, simply make the default disk (and drive) the same as that which contain these files, and type the following command :

SASIM [drive:\data directory] followed by [ENTER].

So if "SASIM C:\DATA" is typed in, the data would be read from and written to the directory "C:\DATA"

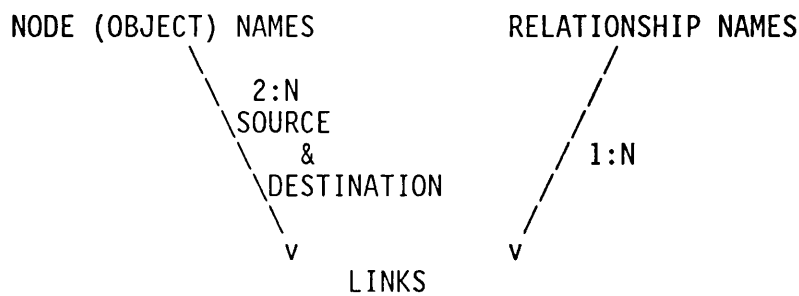
A second parameter could be a screen mode number (as defined in TURBO 4 manual). This is not normally necessary as SASIM automatically detects the type of screen.

APPENDIX 6

KNOWLEDGE REPRESENTATION IN MEMORY

In order to achieve a theoretically sound memory representation, I relied on Network Database Theory to model the relationships. The main analogy is that each of the three main linked lists in memory correspond to a database table.

The network data model is as follows :



Each of the tables have the following fields :

1) Node (Object) Names :

FIELD NAMES	TYPE
NAME	Character (LENGTH 20)
INDEX	Pointer to a list of relationships. The list is of all relationships which this object is involved in.

2) Relationship Names :

FIELD NAMES	TYPE
NAME	Character (LENGTH 20)
INDEX	Pointer to a list of all links which this relationship is involved in.

3) Links :

FIELD NAMES	TYPE
SOURCE	Pointer to the source object of this link.
DESTINATION	Pointer to the destination object of this link.
RELATIONSHIP	Pointer to the relationship of this link

N.B.

- 1) All the records in a table are linked together with a 'forward' pointer. If the records were stored in a memory array, these pointers are unnecessary, since the record number implies the next record, but this would seriously limit the flexibility of using records and pointers.
- 2) All the lists are implemented as pointer chains, terminated by a 'NIL' pointer, as opposed to a circular list which is asymmetrical when the list is empty.
- 3) The Objects table is also sorted alphabetically on the name to aid both the speed of a 'find' enquiry, as well as to make the order of any query output more sensible.
- 4) The reason for this theoretical background is that if a relationship or object was renamed, only one field has to be changed (but the object list needs to be resorted to maintain the alphabetic listing). This is a direct result of the elimination of redundancy in the data.
- 5) Relational Database advocates would have replaced all pointers by 'name' fields, implying the need for indexes to access a record given a name. This would considerably slow down access speeds, as well as increasing the programming and memory overheads.

APPENDIX 7

GLOSSARY OF ABBREVIATIONS / NAMES

ACD	Activity Cycle Diagram. A very widely used modelling method for representing the logic of a simulation problem. Popularized by Hills (1971).
AI	Artificial intelligence.
ASPES	A Skeletal Pascal Expert System. Developed at the LSE [Doukidis and Paul, 1987].
AUTOSIM	An Integrated Simulation Program Generator developed at LSE [Paul and Chew, 1987] [Chew et al, 1985].
BRANCH AND BOUND	A search algorithm which guarantees optimality.
BREADTH FIRST	A search algorithm which tries to explore nearest links first.
CAPS	An early Interactive Simulation Program Generator for ECSL. Uses an ACD as the main input. Written by Clementson (1982).
CASM	The Computer Aided Simulation Model. A team set up in 1982 at the London School of Economics to investigate computer aided simulation modelling [Balmer and Paul, 1986].
CSL	Control and Simulation Language. Developed by Buxton and Laski (1962) at IBM UK and Esso.
DEPTH FIRST	A search algorithm which tries to explore long links first.
DEVS	Discrete event system specification formalism. A hierarchical, modular formalism [Kim and Zeigler, 1987].
ECSL	Extended Control and simulation Language. An extension of CSL. Written by Clementson (1982). Uses an activity scan.
eLSE	Extended Lancaster Simulation Environment. Simulation routines originally created at Lancaster University and subsequently modified at the LSE [Crookes et al, 1986]. After 1987 upgrade, it is now called LIBSIM.
GASP IV	Simulation language combining discrete change, continuous change and mixed models. [Pritsker, 1974].

HOCUS	Simulation package. Uses an ACD as an input [Syzmankiewicz, 1984].
ISPG	Interactive Simulation Program Generator.
KBS	The Knowledge-based Simulation System. [Baskaran and Reddy, 1984] [Reddy et al, 1986]. renamed Simulation Craft.
LANGEN	Renamed to AUTOSIM during 1986.
LIBSIM	Latest version of eLSE.
LSE	London School of Economics.
NLUP	Natural Language Understanding and Processing.
ROSS	Rand Object-oriented Simulation System. An english-like, interactive object-orientated language implemented in LISP. [McArthur et al, 1986].
SASIM	Semantic Analyser in Simulation Methodology. Program implementing semantic modelling methodology [Barakat and Paul, 1988].
SEMANTIC NETWORKS	A graphical AI knowledge-base. It is described in Rich (1983).
SIMKIT	An object-orientated simulation language. Produced by Intellicorp. Requires KEE (the Knowledge Engineering Environment) running on a dedicated Hi-resolution graphics Lisp machine [Harmon and King, 1985].
SIMSCRIPT	A very popular commercial simulation language [Caci, 1976].
SIMULA	The first object orientated language which was initially designed as a discrete event simulation language [Birtwistle et al, 1979].
SIPDES	A Simulation Program Debugger using an Expert System. [Doukidis and Paul, 1991]. Developed using the ASPES shell [Doukidis and Paul, 1987].
SMDE	Simulation Model Development Prototype. Developed as a research prototype by Balci and Nance (1987).
SPIF	Simulation Problem Intelligent Formulator. Program develop by Dr George Doukidis, which attempts natural language processing for a simulation. [Doukidis, 1985].

TESS The extended simulation system [Stanbridge et al, 1985].

T-PROLOG An object-oriented, Prolog based simulation language. A combined discrete/continuous version is available, called TC-Prolog. Developed by Futo and his associates at the Institute for Coordination of Computer Techniques in Hungary. [Futo and Szeredi, 1982].

TURBO A very good Pascal compiler for an IBM PC.

VAX DEC mini computer.

VS6 An advanced simulation program environment developed at LSE [Knox, 1988].

WFF Well-Formed Formulas, used in predicate logic knowledge-bases.

APPENDIX 8

BIBLIOGRAPHY

Ackoff, R.L. and Sasienei, M.W. (1968). "Fundamentals of Operation Research". Wiley, New York.

Au, G. and Paul, R.G. (1990). "A Complete Graphical Discrete Event Simulation Environment". CASM Report, Department of Information Systems, London School of Economics and Political Science, University of London.

Balci (1986). "Requirements for model development environments." Computer Operational Research 13, 53-67.

Balci, O. and Nance, R.E. (1987). "Simulation Model Development Environments : A Research Prototype". Journal of the Operational Research Society 38, 753-763.

Balci, O. and Nance, R.E. (1987b). "Simulation Support : prototyping the automation-based paradigm". In proceedings of the 1987 Winter Simulation Conference, Atlanta, Georgia, December 1987, 478-485.

Balmer, D.W. (1987). "Modelling styles and their support in the CASM environment". In proceedings of the 1987 Winter Simulation Conference, Atlanta, Georgia, December 1987, 478-485.

Balmer, D.W. and Paul, R.J. (1986). "CASM - the right environment for simulation". Journal of the Operational Research Society 37, 443-452.

Barakat, M. and Paul, R.J. (1988). "Semantic Modelling for Discrete Event Simulation". CASM Report, Department of Information Systems, London School of Economics and Political Science, University of London.

Birtwistle, G.M.; Dahl, O.J.; Myhrhaug, B. and Nygaard, K. (1979). "SIMULA Begin". Chartwell-Brant, Bromley, England.

Birtwistle, J.J. and Wyvill, B.L.M. (1984). "ANDES : an environment for animated discrete event simulation". United Kingdom Simulation Conference, Bath, September 1984.

Browston, L.; Farrell, R.; Kant, E. and Martin, N. (1985). "Programming Expert Systems in OPS5-Introduction to Rule-Based Programming". Addison-Wesley, Reading, Massachusetts.

Buxton, J.N. and Laski, J.G. (1962). "Control and simulation language". A description of CSL. The Computer Journal, 5,3.

Caci Inc (1976). "Simscrip II.5 Reference Handbook". CACI Inc, Los Angeles.

- Carnegie Group (1986). "A natural language interface to a manufacturing simulation expert system". Carnegie Group, Pittsburgh.
- Ceric, V. and Paul, R.J. (1989). "Preliminary Investigations into simulation model representation". 11th International Symposium "Computer at the University", Cavtat, 1989.
- Chapman, M. and Dayer-Smith, C. (1990). "VS7: A comprehensive Product Review". Prospect Group, Kent.
- Chew, S.T.; Paul, R.J. and Balmer, D.W. (1985). "Three Phase Simulation Modelling using An Interactive Simulation Program Generator". CASM Report, Department of Statistical and Mathematical Science, London School of Economics and Political Science, University of London.
- Chomsky, N. (1957). "Syntactic Structures". La Haye, Mouton.
- Clementson, A.T. (1982). "Extended Control and Simulation Language". Cle. Com. Ltd, Birmingham.
- Crookes, J.G.; Balmer, D.W; Chew, S.T. and Paul, R.J. (1986). "A Three Phase Simulation Modelling System Written in Pascal". Journal of the Operational Research Society 37, 603-618.
- Crookes, J.G. (1987). "Generators, Generic Models and Methodology". Journal of the Operational Research Society 38, 765-768.
- Domingo, L. (1991). "Formal Methods of Specifying Discrete Event Simulation Models". PhD Thesis, London School of Economics and Political Science, University of London.
- Doukidis, G.I. (1985). "Discrete Event Simulation Model Formulation using Natural Language Understanding Systems". PhD Thesis, London School of Economics and Political Science, University of London.
- Doukidis, G.I. (1987). "On the Homology of Simulation with AI". Journal of the Operational Research Society 38, 701-712.
- Doukidis, G.I. and Paul, R.J. (1986). "ASPES : A Skeletal Pascal Expert System". Expert Systems and Artificial Intelligence in Decision Support Systems. (H.G. Sol et al, Eds.). Reidel, Dordrecht, 227-246.
- Doukidis, G.I. and Paul, R.J. (1991). "SIPDES : A Simulation Program Debugger using an Expert System". Expert Systems with Application 2.
- El Sheik, A.A.R. 1987. "Simulation modelling using a relational database package". PhD Thesis, University of London.

El Sheik, A.A.R. and Paul, R.J. (1988). "INGRESSIM: simulation modelling using a relational database package INGRES". CASM Research Report, Dept of Statistics, London School of Economics and Political Science, University of London.

El Sheikh, A.A.R.; Paul, R.J.; Harding, A.S. and Balmer, D.W. (1987). "A Microcomputer-Based Simulation Study of a Port". Journal of the Operational Research Society 38, 673-681.

Fishwick, P. (1985). "Hierarchical reasoning: simulating complex processes over multiple levels of abstraction". PhD Thesis, Department of Computer and Information Sciences, University of Pennsylvania.

Flitman A.M. and Hurrión R.D. (1987). "Linking Discrete-Event Simulation Models with Expert Systems". Journal of the Operational Research Society 38, 723-733.

Futo, I. and Szeredi, J. (1982). "A discrete simulation system based on artificial intelligence techniques". In Discrete Simulation and related fields (I. Javor, Ed.), 135-150. North-Holland. Writing about the T-Prolog and TC-Prolog packages.

Goodman, D.H.; Balmer, D.W. and Doukidis, G.I. (1987). "Interfacing expert systems and simulation for job-shop production scheduling". Presented at the 3rd International Expert Systems Conference, London, 1987.

Harmon, P. and King, D. (1985). "Expert Systems, Artificial Intelligence in Business", 218. Wiley, New York.

Heidorn, G. (1972). "Natural language inputs to a simulation programming system". Technical Report, Naval Postgraduate School.

Henriksen, J.O. (1983). "The integrated simulation environment (Simulation software of the 1990s)". Operational Research 31, 1053-1073.

Hills, P.R. (1971). "HOCUS". P-E Group, Egham, Surrey.

Holder, R.D. and Gittins, R.P. (1989). "The Effects of warship and Replenishment Ship attrition on War Arsenal Requirements". Journal of the Operational Research Society 40, 167-175.

Hu, J. (1989). "Knowledge-based design support environment for design automation and performance evaluation". PhD Thesis, University of Arizona, Tucson, Arizona.

Hu, J.; Huang, Y. and Rozenblit (1989). "Frases - A knowledge representation scheme for engineering design". In Advances in AI and Simulation. The Society for Computer Simulation, Simulation Series 20, 141-146.

- Hurrion, R.D. (1976). "The design use and required facilities of an interactive visual computer simulation language to explore production planning problems". PhD Thesis, University of London.
- Hurrion, R.D. (1986). "Visual Interactive Modelling". In International Trends in Manufacturing Technology, Simulation, 3-13. IFS (Publications) Ltd.
- Hurrion, R.D. and Secker, R.J.R. (1978). "Visual interactive simulation, an aid to decision making". Omega, 6(5), 419-426.
- Khoshnevis, B. and Chen, A.P. (1986). "An expert simulation model builder". In Intelligent Simulation Environments, (P. Luker and H. Adelsberger, Eds.). The Society for Computer Simulation, Simulation Series 17, 129-132.
- Kim, T.G. and Zeigler, B.P. (1987). "The DEVS formalism : hierarchical, modular system specification in an object-oriented framework". In proceedings of the 1987 Winter Simulation Conference, Atlanta, Georgia, December 1987, 559-566.
- Kim, T.G. (1988). "A knowledge-based environment for hierarchical modelling and simulation". Technical Report mAIS-7, PhD Thesis, University of Arizona, Tucson, Arizona.
- Klahr, P.; Ellis, J.W.; Giarla, W.D.; Narain, S.; Cesar, E.M. and Turner, S.R. (1986). "TWIRL : tactical warfare in ROSS language". In Expert Systems Techniques, Tools and applications, 224-273. Addison-Wesley, Reading, Massachusetts.
- Knox, P.M. (1988). "Automated Graphically-Based Discrete-Event Simulation Systems". PhD Thesis, London School of Economics and Political Science, University of London.
- Kuipers, B.J. (1975). "A frame for frames". In Representation and Understanding (D.G. Bobrow and A. Collins, Eds.). Academic Press, New York.
- Lehmann, A.; Knodler, B.; Kwee E. and Szczerbicka (1985). "Dialog-orientated and knowledge-based modelling in a typical PC environment". In proceedings of the European Conference on AI applied to Simulation, Ghent, Belgium. The Society for Computer Simulation, Simulation Series 18, 91-96.

Markowitz, H.M. (1979). "SIMSCRIPT: Past, Present and some thoughts for the future". In Current Issues in Computer Simulation (N.R. Adam, and A. Dogramaci, eds.). Academic Press, New York.

Mashhour, A. (1989). "Automated Simulation Program generation Using a Relational Database Simulation System". PhD Thesis, London School of Economics and Political Science, University of London.

Mathewson, S.C. (1974). "Simulation Program Generators". Simulation 23(6), 181-189.

Mathewson, S.C. (1977). "A Programming Manual for SIMON Simulation in FORTRAN". Imperial College, University of London.

Mathewson, S.C. (1982). "DRAFT II/FORTRAN Manual". Department Management Science, Imperial College, University of London.

Mathewson, S.C. (1985). "Simulation program generators: code and animation on a PC". Journal of the Operational Research Society 36, 583-589.

Mathewson, S.C. and Beasley, J.E. (1976). "DRAFT/SIMULA". In proceedings of Fourth SIMULA Users Conference. National Computer Centre.

McArthur, D.J.; Klahr, P. and Narain, S. (1986). "ROSS : an object-oriented language for constructing simulations". In Expert Systems : Techniques, Tools and Applications (P. Klahr and D.A. Waterman, Eds.), 70-91 Addison-Wesley, Reading, Massachusetts.

Middleton, S. and Zanconato, R. (1986). "BLOBS: an object orientated language for simulation and reasoning". In Artificial Intelligence applied to Simulation (E.J. Kerckhoffs, G.C. Vansteenkiste, and B.P. Zeigler, Eds.), 130-135, The Society for Computer Simulation, San Diego, California.

Minsky, M. (1975). "A framework for representing knowledge". In The Psychology of Computer Vision (P. Winston, Ed.). McGraw-Hill, New York.

Murray, K.J. and Sheppard, S.V. (1987). "Automatic model synthesis using automatic programming and expert systems techniques towards simulation modelling". In proceedings of the 1987 Winter Simulation Conference, Atlanta, Georgia, December 1987, 534-543.

Nance, R.E. (1981). "Model representation in discrete event simulation: the conical methodology". Technical report CS81003-R, Department of Computer Science, Virginia Tech, Blacksburg, Virginia.

Nance, R.E. (1983). "A tutorial view of simulation model development". In proceedings of the 1983 Winter Simulation Conference, Arlington, Va, December 1983, 325-331.

Nance, R.E.; Balci O. and Moose, R.L. (1984). "Evaluation of the UNIX host for a model development environment". In proceedings of the 1984 Winter Simulation Conference, Dallas, Texas, December 1984, 577-584.

Newell, A. and Simon, H.A. (1972). "Human Problem Solving". Prentice-Hall, Englewood Cliffs, New Jersey.

O'Keefe, R.M. (1984). "Developing simulation models: An interpreter for visual interactive simulation". PhD Thesis, University of Southampton.

O'Keefe, R.M. (1986a). "Advisory systems in simulation". In AI Applied to Simulation, (E.J. Kerckhoffs, G.C. Vansteenkiste, and B.P. Zeigler, Eds.). The Society for Computer Simulation, Simulation Series 18, 73-78.

O'Keefe, R.M. (1986b). "Simulation and expert systems: a taxonomy and some examples". Simulation 46, 10-16.

O'Keefe, R.M. and Roach, J.W. (1987). "AI Approaches to simulation". Journal of the Operational Research Society 38, 713-722.

Oldfather, P.; Ginsberg, A.S.; Love, P.L. and Markowitz, H.M. (1966). "Programming by Questionnaire : How to construct a Program Generator". RAND report RM-5129-PR.

Oren, T.I. and Aytac, K.Z. (1985). "Architecture of MAGEST: a knowledge-based modelling and simulation system". In Simulation in Research and Development. The Society for Computer Simulation, Simulation Series 17, 99-109.

Paul, R.J. (1987). "Editorial". Journal of the Operational Research Society 38, 671-672.

Paul, R.J. (1988). "Simulation modelling : The CASM project". Paper presented at The Annual Operational Research Symposium of Yugoslavia (1988) and The 2nd Brazilian Workshop on Simulation (1988).

Paul, R.J. (1989a). "Artificial intelligence and simulation modelling". In Computer Modelling for Discrete Event Simulation (M. Pidd, Ed.), Wiley, Chichester.

Paul, R.J. (1989b). "Combining artificial intelligence and simulation". In Computer Modelling for Discrete Event Simulation (M. Pidd, Ed.). Wiley, Chichester.

Paul, R.J. (1991). "Recent developments in simulation modelling". Journal of the Operational Research Society 42, 217-226.

Paul, R.J. and Balmer, D.W. (1991). "Simulation Modelling". Chartwell-Bratt Student Text Series.

- Paul, R.J. and Chew, S.T. (1987). "An interactive Simulation Program Generator". *Journal of the Operational Research Society* 38, 735-752.
- Paul, R.J. and Ceric, V. (1990). "Conceptual modelling in discrete event simulation using diagrammatic representation". London School of Economics and Political Science (University of London) and University of Zagreb joint paper.
- Pidd (1987). "Simulating automated food plants". *Journal of the Operational Research Society* 38, 683-692.
- Pidd, M. (1988). "Computer simulation in management science". Wiley, Chichester. 2nd edition.
- Post, E. (1943). "Formal reductions of the general combinatorial decision problem". *American Journal of Mathematics*. 65, 197-268.
- Pritsker, A.A.B. (1974). "The GASP IV Simulation Language". Wiley, New York.
- Quillian, R. (1968). "Semantic memory". In *Semantic Information Processing* (M. Minsky, Ed.). MIT Press, Cambridge, Massachusetts.
- Raphael, B. (1968). "A computer program for semantic information retrieval". In *Semantic Information Processing* (M. Minsky, Ed.). MIT Press, Cambridge, Massachusetts.
- Reddy, Y.V.; Fox, M.S.; Husain, N. and McRoberts, M. (1986). "The knowledge-based simulation system". *IEEE Software* 3, 26-37.
- Reddy, Y.V. and Fox, M.S. (1982). "KBS : an artificial intelligence approach to flexible simulation". *IEEE Software* 3, 26-37.
- Reese, R. and Sheppard, S. (1983). "A software development environment for simulation programming". In *proceedings of the 1983 Winter Simulation Conference*, Arlington, Va, December 1983, 419-426.
- Rich (1983). "Artificial Intelligence", McGraw-Hill, New York.
- Rivett, B.H.P. (1972). "The Art of Model Building". Wiley, Chichester.
- Robertson, P. (1986). "A rule based expert simulation environment". In *proceedings of the conference on Intelligent Simulation Environments*, San Diego, California. The Society for Computer Simulation, Simulation Series 17, 9-15.
- Rozenblit, J.W. (1985). "A conceptual basis for integrated model-based system design". PhD Thesis, Wayne State University, Detroit, Michigan.

Rozenblit, J.W.; Hu, J.; Kim, T.G. and Zeigler, B.P. (1990). "Knowledge-based Design and Simulation Environment (KBDSE) : Foundational Concepts and Implementation". Journal of the Operational Research Society 41, 475-489.

Rozenblit, J.W. and Zeigler, B.P. (1985). "Concepts for knowledge-based system design environments". In proceedings of the 1985 Winter Simulation Conference, San Francisco, California.

Ruiz-Mier, S.; Talavage, J. and Ben-Arieh, D. (1985). "Towards a knowledge-based network simulation environment". In proceedings of the 1985 Winter Simulation Conference, San Francisco, California, December 1985, 232-236.

Shannon R.E. (1985). "Expert systems and simulation". Simulation 44, 275-293.

Shirai Y. and Tsujii J. (1984). "Artificial intelligence - Concepts, Techniques and Applications". Wiley, Chichester.

Stanbridge, C.R.; Vaughan, D.K. and Sale, M.L. (1985). "A tutorial on TESS : the extended simulation system". In proceedings of the 1985 Winter Simulation Conference, San Francisco, California, December 1985, 73-79.

Sutton, D.W. and Coates, P.A. (1981). "On-line mixture calculation system for stainless steel production by BSC stainless: the least through cost mix system (LTCM)". Journal of the Operational Research Society 32, 165-172

Szymankiewicz, J. (1984). "A Description of the HOCUS simulation system". P-E Information Systems, Egham, Surrey.

Szymankiewicz, J.; Mc Donald J. and Turner, K. (1988). "Solving Business Problems by Simulation". (2nd Edition), McGraw-Hill, Maidenhead.

Taylor, R.P. and Hurrion, R.D. (1988). "An expert advisor for simulation experimental design and analysis". In proceedings of the conference on Artificial Intelligence and Simulation : the diversity of applications (T. Hendson, Ed.). The Society for Computer Simulation, San Diego, California.

Tocher, K.D. (1963). "The Art of Simulation". Hodder and Stoughton Educational publishers, London.

Unger, B.; Dewar, A.; Cleary, J. and Birtwistle (1986). "A distributed software prototyping and simulation environment : Jade". In proceedings of the conference on Intelligent Simulation Environments, San Diego, California. The Society for Computer Simulation, Simulation Series 17, 63-71.

- Vaucher, J.G. (1985). "Views of modelling: comparing the simulation and AI approaches". In 'AI Graphics and Simulation' (G. Birtwistle, Ed.), 3-7. The Society for Computer Simulation.
- Wales, F.J. and Luker, P.A. (1986). "An environment for discrete event simulation". In proceedings of the conference on Intelligent Simulation Environments, San Diego, California. The Society for Computer Simulation, Simulation Series 17, 58-62.
- White, D.J. (1975). "Decision Methodology". Wiley, Chichester.
- Withers, S.J. and Hurrion, R.D. (1982). "The interactive development of visual simulation models". Journal of the Operational Research Society 33, 973-976.
- Zeigler, B.P. (1987). "Hierarchical, modular, discrete-event models in an object-orientated environment". Simulation, 50, 219-230.
- Zeigler, B.P. (1990). "Object-Orientated Simulation with Hierarchical Modular Models: Intelligent Agents and Endomorphic Systems". Academic Press, Boston.
- Zeigler, B.P. and Wael, L. De (1985). "Towards a knowledge based implementation of multifaceted modelling methodology". In proceedings of the European Conference on AI applied to Simulation, Ghent, Belgium. The Society for Computer Simulation, Simulation Series 18, 42-51.