Parallel Processing of Frame-Based Networks

by

Mohammed Hashem Saeedi MSc

A thesis submitted in partial fulfilment of the requirements of the council for National Academic Awards for the degree of Doctor of Philosophy.

Sponsoring Establishment : School of Computing and Management Sciences, Sheffield Hallam University

April 1993

ABSTRACT

This Project involved the development of a simulation of a rectangular array of Processing Elements (PE's), with a dedicated frame based knowledge representation language. The main objective of the Project was to analyse and quantify the gain in speed of execution in a parallel environment, as compared with serial processing.

The computational model of the language consisted of two main components: the knowledge base, and the replicated/distributed inference engine. The knowledge base was assumed to represent real world knowledge, in that it consisted of a large volume of information, which was divided into domains and hierarchies.

When a query is made, appropriate portions of the knowledge base are mapped to the array of PEs on a one-to-one basis (one frame/PE), where each PE is capable of performing any relevant operations itself.

The execution of a query is based on the propagation of messages across the array of PEs, where each message is contained in a data packet. Each packet holds the query-frame, created by interacting with the user, together with other relevant information used for knowledge manipulation.

The main inference mechanism in the system is based on the parallel inheritance of properties, where each data packet carries inherited data from higher level to lower level frames, within the appropriate hierarchies. As each packet arrives at a PE which contains a relevant frame, a series of matching, and consequently, inheritance operations are performed.

An algorithm, superimposed at the highest level of the system, computes time delays in relation to the overall architecture of the machine. There are two main operations for which time penalties are calculated : frame-processing and communication. The frame processing involves matching and inheritance operations, and the communication operation involves message passing and data packet traversal.

During each execution cycle, the time penalties for both processing and communication are computed and stored in a file. These files are then used by a graphics package which transforms the numerical data into a set of graphs. These graphs are utilised in the analysis of the behaviour of the simulation. The analysis of the test-runs, and of their associated graphs, has yielded positive and encouraging results, demonstrating that there can be an average of a 35 fold gain in the speed of execution.

ACKNOWLEDGEMENTS

This project was financed by ALVEY/SERC research grant GR/D 40081 - IKBS/ARCH/068/131 with ICL plc as an "industrial uncle".

I would like to thank my first and second supervisors : John Brown and Ian Morrey, for all their help and advice. I would also like to thank my dear wife who has been a great support throughout the lifetime of this project.

TABLE OF CONTENTS

TABLE OF FIGURES

1.0 CHAPTER ONE : INTRODUCTION

1.1 INTRODUCTION

We have come a long way in the history of mankind in terms of achievement in technological, sociological, political and many other aspects of life. These achievements are a result of man's endeavour from the ancient to the present time, and due to his innovation and creativity. In the East, people used the abacus as a mechanical calculator for their commercial applications (Metropolis 1980), whereas in the West, it was only after the introduction of the Arabic numeral system that the medieval European was able to perform complicated calculations. It was around the 16th century that, instead of using empirical knowledge, mariners used mathematically based charts to find their destinations (Pratt 1987). After the renaissance and the establishment of modern science, the idea that thinking might be provided by a machine, was born. This idea was later put in practice -albeit in a limited form- by people including Leibniz's calculator and the Analytical Engine created by Babbage (Hyman 1991). Zuse in Berlin in 1936, Atanasoff in Ohio in 1937, and the Bell Telephone laboratories in New York (Schutzer 1987), and many other individuals and organisations, were theorising on, and in some cases, developing, new calculating machines. During the second world war, and the dire necessities associated with it, machines, mostly electromechanically based, were developed for tasks including deciphering radio traffic, and the calculation of firing tables for artillery. This led to the development of machines such as ENIAC in USA and ENIGMA in Britain (Metropolis 1980). Associated with the rapid development of machines in this century, was "electricity". The crucial role that electricity played was to replace the heavy mechanical components (eg metal rods and cogs etc.) with cables and switches (Andriole 1985) and thus, a substantial increase in the speed of processing was gained.

Although the automation of the mind has long been an ambition for man, all the efforts up to the beginning of this century were 'only' leading to automation of calculation ie, the development of machines for faster mathematical computation, namely addition, subtraction, division and multiplication. It was only later, with the tremendous advancements in mathematics, electrical and electronic engineering, psychology and philosophy that the concept of automation of thought was perceived, by pioneers like Alan Turing, Emil Post and Alonzo Church, to be a possibility (Cohen 1981). Turing's idea was to build, not an automated reasoner, nor a machine capable of understanding the universal language of algebra, but a machine that would have powers coextensive with that of human brain. This idea was shared by many pioneers involved with automated computational theories, including John von Neumann, but his perspective

was influenced by the technological limitations imposed on developing such machines. He took charge of the next development of a new machine based on ENIAC, called EDVAC. In this machine, a new concept of fundamental importance was developed[1], that of the "stored program" (Albus 1981).

The development of digital computers continued, and along with it, the desire to imitate human thinking grew. In this development, the mathematicians had the most important role, and their main interest was to develop a machine which was able to make a contribution to mathematics. Another group: engineers, were interested in modern computers and, at a practical level, understood them well. The concept of automatic data-handling was an attractive idea for people in commerce, and those who were interested in non-mathematical applications in computing. There were others, who had the intention of utilising computers for intellectual purposes. At the beginning of the twentieth century, a certain amount of research was done on the nervous system in the context of psychology; in particular, behaviourism. This field of research became known as "cybernetics".

The members of the cybernetics group were from different backgrounds and disciplines, including mathematicians and engineers, whose programs were drawn together under the inspiration of Weiner in 1940s. The main research projects in cybernetics, then, was concerened with the application of control engineering concepts to the understanding of physiological and neurophysiological processes. This work involved people like Weiner, Rosenblueth, McCulloch, Pitts and others (Pratt 1987). With the observations made by Cajal, through his work involving anatomy of the nervous system (Rumelhart 1987), the cybernetics group were able to theorise on the concepts and properties that he produced. As a result of Cajal's work, it was possible to work on systems made up of a number of neurons, and use their interconnections and the properties they offered. The interconnections of neurons and specified properties attributed to each neuron were seen as nerve-nets (or neural net of today), and a substantial amount of work was done by people like McCulloch and Pitts, which greatly contributed to today's understanding of neural nets (Aleksander 1990).

In Dartmouth college, in 1956, a conference was organised by a young mathematician, John McCarthy and his colleague Marvin Minsky from MIT. In this conference McCarthy proposed a study of Artificial Intelligence which would describe the creation of a machine that will simulate human's intelligence (Charniak 1987). Since 1956, the term "Artificial Intelligence" ("AI") has been used for every aspect of developing

---

[1] Note that there is a controversy about the origin of this idea of " stored program". For further details see (Pratt 1987), pp167.

systems that had something to do with human thought processing. Visual perception and pattern recognition, natural language understanding, problem solving and game playing, are some of the fields which were rapidly developing under AI's umbrella. It has been claimed that modern perspectives in psychology, embodied in congnitive psychology, owes its existence and reputation to AI (Winston 1977). Newell and Simon (Newell 1972, Newell 1976) were two of the pioneers that brought this new perspective to psychology, and exerted a strong influence both in this field and in other subfields of AI.

After developing machines with stored programs however, there were opportunities for people working in the AI community, to implement different applications. But while the complexities of these applications increased, their requirements remained the same; that is, an embodied program, a large amount of memory, and high speed.

Since 1950, in the succeeding decades, the complexity and consequently the widening of applications in AI, made it apparent that the serial machines, or von Neumann machines, could not meet their requirements. The main characteristic of these applications is the amount of knowledge that they require. Given the time constraints, it would be impossible to explore the information, which is a pre-requisite to any consequent inferences and conclusions. The realisation of this concept was the main encouragement for a big international push, that started in 1981. At this time, large computing research and development projects had been started by almost two dozen industrial nations. EEC's ESPRIT (European Strategy Research Program in Information Technology), UK's ALVEY, Japan's ICOT (Institute for New Generation Computer Technology), MCC (Computer Technology Corporation) and DARPA (Defence Advance Research Project Agency) in the USA and national programs in Soviet Union and other Eastern Bloc Nations, geared up to pursue advanced computing technology and developing machines for diverse AI applications (A datamation Staff Report 1985, Delgado-Frias 1987). Despite the substantial decrease in financial support for the continuation of this research and development, all these initiatives pushed the state of the art forward in many directions, covering a wide range of branches in AI and in computer science generally. Some of these applications are :

a) Knowledge representation paradigm,
b) Knowledge based systems,
c) Natural language understanding and speech processing,
d) Robotics[2],
e) Theorem proving,

---

2 This is not a common view. This field involves AI, electrical, mechanical and optical engineering.

f) Automatic programming,

g) Perception.

The first two application areas of AI, knowledge representation and knowledge based systems (eg, expert systems), are undoubtedly the most important areas that have influenced computers, their architectures and languages. In many automated problem solving systems (eg expert systems), there are three main sections : the knowledge base, that is the representation of empirical human knowledge in a specific domain; the inference engine, employed to deduce facts or induce hypotheses in the given domain; and the interface between the outside world and the system.

In order to automate any problem solving, there are two main principles that have to be considered; searching, and knowledge representation. The representation of knowledge involves specific formatting of the information retrieved from the application domain and their relevant expert(s), which should be represented correctly. Representation should be simple and easy to understand, so that it can be modified and updated. A bad representation will always produce difficulties in its manipulation (eg, Roman numbering systems).

The methods of searching or exploring the knowledge representing the application domain, are also extremely important. A method should, given a number of options, indicate which is to be selected in order to reach the goal. The problem of search is to find the next appropriate move (like chess). In a search space which encapsulates the problem domain, we start from a node (see figure 1.1) and try to get to the goal state.

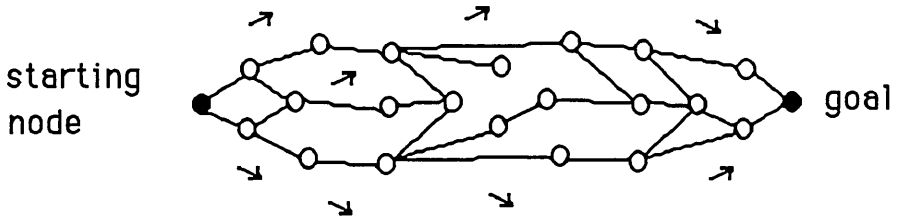


Figure 1.1, a graphical representation of a search space.

In the process of searching, each time that a move is taken, there is a new series of options available for selection. This causes a problem, in particular in a complex search space, in that the number of options become so great that they cannot all, within a given time constraint, be exhaustively investigated. This is called the combinatorial explosion. In AI, many attempts have been made to find the most economical way of arriving at the goal state (Kanal 1988). It must be noted that in a complex problem, because of the great number of options available, even finding the most relatively economical option is still a challenge. As an example, in order to find the shortest route

between two towns in an 'n' town map, there are (n - 1)! possibilities to explore (say 10 towns has 9! which is 362880 possibilities).

It is apparent therefore, that a von Neumann machine by performing serial operations will run out of time while it is exhaustively exploring all the possibilities given in a large search space. One way to tackle this problem is to reduce the size of the application domain, and consequently the search space, by which the system will be applied to a relatively small area. As mentioned above, in order to reduce the possible occurrence of combinatorial explosion, further guidance can be added to the searching algorithm based on heuristics, and trial and error. This concept has been widely utilised in today's expert systems. There is another option which is the essence of this thesis; that is, parallelism. In an ideal parallel environment, the whole volume of the relevant knowledge is explored in parallel without exceeding the limitation imposed by the time constraint.

The other principle to be considered in constructing a system to solve problems, is knowledge. We spend most of our lives solving problems; crossing the road, playing games, even talking to each other, can be regarded as problem solving. But knowledge is of utmost importance in the process of solving problems. We need to have knowledge as the background to the problem, knowledge on how to solve the problem, knowledge on making relevant conclusions, and knowledge and the ability for, if necessary, adding to the existing knowledge, which constitutes learning.

Knowledge can be divided into three groups: ordinary knowledge, expertise knowledge, and common sense knowledge (Alty 1990). To solve problems associated with ordinary knowledge, there are many traditional computer languages that have already been developed (ie 4GL's etc). Expertise knowledge can be used in a small domain for solving problems (expert systems). The problems related to common sense knowledge are extremely difficult to solve. This is due to the volume of the knowledge which has been acquired through the lifetime of an individual under different conditions and situations. Consequently, the sheer size of common sense knowledge disables today's serial machines to explore it within an appropriate time.

It is a general consensus that in many AI applications, such as speech recognition, vision or some higher level tasks like medical diagnosis and problem solving in general, the need for much faster machines is apparent. This is directly due to the amount of information that has to be processed. Operations such as retrieval, exploration and manipulation of knowledge, and consequent inferences for example, necessitate a great deal of computational power, that conventional von Neumann machines have been proven to offer inadequately. In reducing the processing time

taken by operations that are required in the frame work of today's AI applications, a fundamental departure from the traditional computer systems has occurred. This goes hand in hand with the rapid advancement in hardware technology.

The pace of technological advancement in computer science and its related fields has been so fast that 50 years history of this field can be equivalent to centuries of development in other fields. In the very near future, cathode ray tubes are going to be replaced with large liquid crystal screens (Foremski 1990). It has been claimed that the holographic memory systems are providing enormous potential for data storage and retrieval, much faster than any existing media (Bains 1990). At the same time there is substantial development in disk technology, with access speeds ranging from 300Kb to 6Mb per second (Classe 1990). The micro chip industry is continuously providing new and faster chips, to the extent that Intel has set up a goal that by the year 2000, it will produce a chip running at 250MHz (Hayward 1991). In the early days of electronics, circuits were constructed from large individual components such as capacitors, resistors, inductors and valves, which were mounted on some kind of metal chassis and hand wired (Metropolis 1980). Today, there are as many as one million transistors on a semiconductor material and it has been estimated that the number of transistors on a single chip will be increased to 100 million (Robbins 1990).

It is neither the function of this report, nor is it possible here, to list the advancements of technology in computer science, but a comparison between today's supercomputers and that of the first valve-based computers reveals an astonishing advancement. The operations involved with the calculation of supersonic airflow would take the latest Cray supercomputer little more than half an hour; whereas in contrast, it has been estimated that it would have taken ENIAC around 27 years to perform the same calculation (Roche 1990). In other words, today's computers are nearly a quarter of million times faster that of the earliest models, built less than 50 years ago.

Along with this advancement in hardware technology, the software technology has been improved extensively. This development inevitably lags behind hardware design, nevertheless, any computer requires a variety of programs to make it useful. In the beginning, there was machine language, which involved programming in binary operations. Assembly language was the first step towards a better and friendlier environment for the users. This development continued and as the result, better and much friendlier languages were developed. At present, attempts have been made to develop new languages that will provide facilities for both novice and experienced users. The hope is to completely hide the complexities of these languages/environments from users who have no, or very little, understanding of computers. These systems,

will therefore, have to provide environments which are intelligent and can make decisions upon the interaction with the user, and their queries. Intelligent knowledge based systems is a general term used for systems that have attempted to incorporate such an environment.

To provide an intelligent environment which can deal with problems such as those mentioned above, or those that are based on common sense, we need a large volume of data and a fast machine to explore it within a given time-band. Note that, the concept of fast machines is not merely computers with micro processors having high speed clocks, but those machines that can explore a network or group of trees of information in a given time-band. In spite of an extremely fast processing speed, von Neumann machines will have to search these search spaces serially. The sheer size of the knowledge to be explored causes a bottle-neck, in particular bearing in mind the concept of combinatorial explosion.

In knowledge base systems, one of the major problems with serial execution, is the validity of the system. That is, since all the paths available cannot be taken, how can we be sure that one of the unidentified paths, if taken, will not return an unexpected result ? This concept in a parallel environment should not exist where all the paths will be explored in parallel.

An ideal AI machine must be able to explore large multi-domain knowledge bases in parallel, and rapidly make appropriate inferences. This, in turn, should exhibit intelligence or perform intelligent behaviour. Thus in a parallel machine, there are two main principles to consider; the method of representing knowledge and the topology of the Processing Elements ("PEs"), that constitute the parallel machine.

In the following chapters the issues discussed above are addressed in more detail. This involves different methods for knowledge representation, different parallel architectures, methods of exploring the knowledge bases and their parallel executions.

In chapter 2, the three major models of representing knowledge; semantic networks, frame-based networks and production systems, are discussed. In chapter 3, a brief review of various parallel architectures is made. Later in this chapter, the taxonomy of AI machines, mainly knowledge based machines, followed by the architectural structure of the Sheffield Machine ("SM") are discussed. The chapter ends with a review and comparisons of various parallel architectures, which have certain characteristics relevant to the SM. In chapter 4, the SM's knowledge representation language, which includes the knowledge base and its control mechanism are discussed. The discussion includes the frames (and their structures) in the SM's knowledge base,

the queries available in the system, parallel propagation, parallel inheritance and inferences, in the SM. The main emphasis in this chapter is on the distributed/replicated control mechanism that includes methods employed for interrogating the knowledge base, where parallel propagation and parallel inheritance are involved. In chapter 5, the simulation of the SM its objectives and operations, its components with their operations and, the program components and data structures developed for creating the simulation program are discussed. The operations of the simulation include querying the system, which in turn involves the benchmark knowledge base and Application Development Tool ("ADT"). Later in the chapter, disk-unit, hashing, retrieval and mapping operations and the rectangular array of PEs are examined. In chapter 6, benchmarking and their evaluation are presented. This involves a discussion on the algorithm and criteria for calculating the time delay of communication and processing operations, the test-runs, their verification and analysis by the graphics package, and concluding remarks. Chapter 7 contains the conclusion and discusses areas for future work. There are 4 appendices; appendix A contains some of the programming code for the simulation, appendix B contains the benchmark knowledge base, appendix C contains some of the test-runs of the simulation, and finally, appendix D contains some of the graphical representations of the test-runs.

1.2 PROJECT OBJECTIVES

At Sheffield City Polytechnic there already exists a computer simulation of a parallel machine; the SM, consisting of a rectangular array of a large number of PEs, written in Pascal. The overall aim of this project was to modify this simulation so that a suitable knowledge representation language could be developed and adopted. Further, by exhaustive testing of the simulation, recommendations will be made as to the optimum scale and technology of the machine, and to predict the likely performance and comparison with alternative serial operations.

Therefore, the detailed objectives in chronological order, are as follows :

i) To perform a literature survey, so that, one or more methods of storing and querying knowledge in the given architecture can be identified as being appropriate.

ii) To evaluate the superficial advantages and disadvantages of each method.

iii) To modify the given simulation to represent a subset of these methods, chosen as superficially successful in the previous stage.

iv) To test and validate the simulations and so develop accurate representations of the real systems.

v) To produce test results to enable quantitative performance comparisons to be made between the various methods studied, and to provide a measure of performance of one or more of these methods, in the context of a typical expert system application.

vi) In parallel with the five objectives just described, keep abreast of research developments elsewhere in parallel architectures and knowledge representation languages, so that continuous comparisons can be made with the SM.

2.0 CHAPTER TWO : THE KNOWLEDGE REPRESENTATION PARADIGM

2.1 INTRODUCTION

Every programming language contains knowledge about the problem that it is solving. Solving differential equations or updating the company's monthly pay-role for example, require knowledge about the appropriate problem domain and the methods of solving them. In conventional programming languages, this knowledge is integrated with the control mechanism and is represented within the program. An attempt to modify, expand or manipulate this information, will involve a considerable amount of complexity. In most AI related applications, a popular approach is to represent the problem domain as a separate entity, called a knowledge base. The knowledge base can be modified and expanded without disturbing the overall structure of the system and the control mechanism. A dedicated machine can, by exploring and manipulating the knowledge base, reach new conclusions.

As mentioned in chapter 1, a system that contains a knowledge base separate from its control unit and inferencing components is called a "knowledge based system". The knowledge base contains facts, rules, heuristics and procedures.

It is the accepted view of the AI community that knowledge, in large quantities and arranged into usable structures, is an essential ingredient of intelligent behaviour. The knowledge required by a system to discover molecular structures[1] (Gaschnig 1982), or for diagnosing a disease by a medical diagnostic system (Shortliffe 1976), is of utmost importance to that system.

In the late 60's, knowledge and the methods for its representation emerged as a separate area of study. Several different approaches for representing knowledge were developed and have resulted in diverse formalisms that are extensively utilised today.

The prominent approaches to knowledge representation are semantic networks (Attardi 1982, Bic 1984), frame-based representation and object oriented representation (Minsky 1974, Goldberg 1984), scripts (Schank 1977), procedural representation (Jackson 1990), production systems (Davis 1980), logical representation, i.e. first order logic (Wellsch 1984), logic programming (Clocksin 1984, Bratko 1990) and knowledge representation languages (Brachman 1985b). It should be noted that these formalisms and their methodologies are not mutually exclusive as they often impinge

---

[1] Given information about the constituents of the compound and mass spectra.

on one other, but nevertheless, they do form a convenient division for particular applications.

For most AI applications, the choice of representation is difficult, since there is a variety of options, and the selection criteria are not clear. Nevertheless, it is essential to employ an appropriate representation so that a good result can be obtained (Woods 1983). Mathematically based representations, such as predicate calculus are, without a doubt, popular formalisms in the AI community, and provide formal precision and interpretability. At the start of this project however, a decision was made to investigate only the more heuristic representations[2], eg semantic networks, frame-based networks and production systems as being the most appropriate for the architecture of the Sheffield Machine ("SM"). In this chapter therefore, a discussion on three major models of representing knowledge; semantic networks, frame-based networks and production systems, is presented.

NETL (Fahlman 1979), and neural networks are to be discussed in chapter 3. This is because of the integrated nature of parallel architecture and knowledge representation in both systems.

2.2 SEMANTIC NETWORKS

In the brief history of Artificial Intelligence ("AI"), many attempts have been made to mimic the structure and the organisation of perceived human memory. One of the approaches was the development of semantic networks. A semantic network is a hierarchical structure which represents a set of objects classified and sub-classified. Properties which are true of the whole class of objects may be specified once at the higher levels of the hierarchy and inherited at lower levels of the hierarchy by default. It is possible however, that the general attributes may not be true for all sub classes, so that an individual concept can have its own default properties.

The basic building blocks of the network are nodes and arcs. Each arc connects exactly two nodes and has a label. The fact that individual arcs are differently labelled allows extremely complex logical propositions to be expressed by the network (Schubert 1976). Precise restrictions on what an arc may represent directly vary between examples of the semantic network representation (Brachman, 1983b).

---

2 Mathematically based representation was adopted in another project (Jelly 1990).

However, all network schemes are based on the idea of knowledge represented in graph structures, with nodes representing concepts connected by links representing semantic relationships between these concepts.

Hendrix's partitioned networks are one type of semantic network. He claimed that the central idea of partitioning is to allow groups of nodes and arcs to be bundled together into units of spaces (Hendrix 1975).

A form of partitioning which has been adopted in this project will be discussed in detail in chapters 4 and 5. Fahlman developed a novel parallel network and knowledge representation language called NETL which was based on the notion of semantic nets (Fahlman 1979). Winston's Structured Descriptions, is another example of semantic networks (Winston 1975). He developed this system while working in the field of machine learning and he was concerned with the notion; "learning from example". An example of representing an arch, using Structured Descriptions, is shown in figure 2.1.

Quillian and Winston based their knowledge representations on psychological models of memory. There are other semantic network-based representations that employ linguistics models. Case Grammar and Conceptual Dependencies are two examples that are based on linguistic models (Filmore 1966) and (Schank 1972).

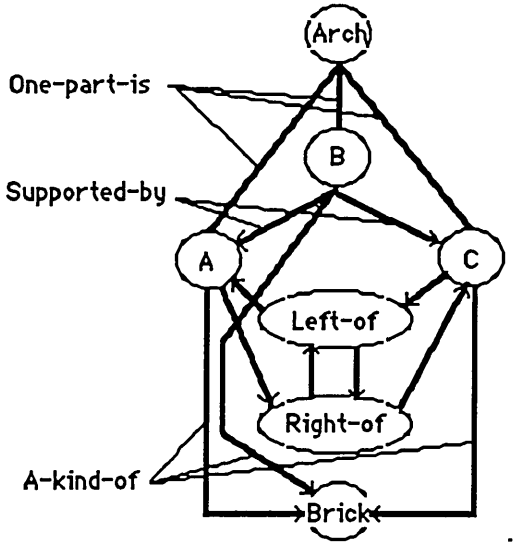


Figure 2.1, Structured Description of an Arch.

It is widely acknowledged that Ross M Quillian proposed the first computational associative network model of semantic memory in his Ph.D thesis in 1966 (Quillian 1968). His network represented a taxonomical hierarchy, in which the basic relationship between objects at two levels consists of a subclass and its superclass, and

contains a description of properties for each class. This gave rise to the possibility of further division of these pairs into sub-properties.

Quillian regarded his network as being capable of inference due to the associative links between any two concepts, and of the inheritance of attributes. The inference technique that he used involved the propagation of activation signals through out the network, to find any intersection between the planes of the two given words. If an intersection was found, then the path from the two given nodes (words) to the point of intersection was seen as the only possible relationship between the two words (Collins 1975, Charniak 1980).

Quillian's work was later developed further by Carbonell, who introduced the notion of instantiation in his SCHOLAR program (Brachman 1978a). In subsequent years, the notion of the semantic network evolved in parallel in different sub-fields of AI, so that, under the name of semantic networks, many different approaches were introduced but no proper formalism was developed. An attempt to rectify this situation, and to encourage the development of a standard formalism for semantic networks, was made by Woods who emphasised several common misinterpretations and misuse in semantic networks representation (Woods 1985). Later KL-ONE, which is a frame-based knowledge representation language, was introduced with a strong emphasis on complex relationships between concepts. The formalism within KL-ONE was based on the SI-nets formalism (Structural Inheritance Networks) developed by Brachman (Brachman 1977a, Brachman 1978a). The endeavour to provide a sound formalism for semantic nets is continuing with some effective results (1986 Touretzky).

Because of the associative links and consequent relationships and paths that are provided within a semantic network, this method of knowledge representation emerges as the primary candidate for a parallel architecture (Feldman 1985, Dixit 1984). There are a number of additional advantages in using this formalism for a parallel architecture. Firstly, the mechanism of inheritance where properties from higher level objects can be inherited by lower level objects in a hierarchy, leads to an economy of implementation. Secondly, the representation of concepts (or objects) and their relationships in a single formalism is an important advantage over formalisms like predicate calculus, where an index is required to define the relationship between any two statements. Thirdly, the property of locality is more pronounced in a semantic net than in any other formalism. Here, locality means the utilisation of the proximate relationship between any two entities. In other words, a cluster of related concepts (objects) can represent the application domain which, will consequently reduce the search space and thus increase the speed of interrogation.

The main inference mechanism in semantic networks is based on the inheritance of properties (Fox 1986). At higher levels of the hierarchy, generic concepts contain general properties that can be inherited at lower levels of the hierarchy. It is through this transition that inference can be made. In Figure 2.2 for example, Human has two legs would imply that Jane as a typical human will have two legs.

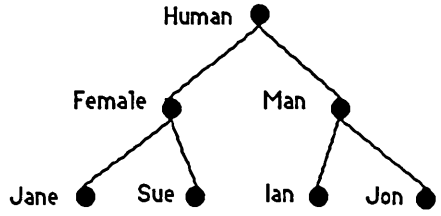


Figure 2.2, showing an example of semantic net.

It is important to notice that, although the general properties and their values can be inherited at lower levels of the hierarchy, an individual concept may have its own values which will override the inherited values. The consideration of exceptions is an important issue in real world knowledge representation. For example, if Sue has lost one of her legs in an accident, the default value for number of legs, which is in this case two, will be overridden by one.

2.3 FRAMES

The concept of Frames originates from Minsky's work on the recognition of objects in vision (Minsky 1974). It did not take long to gain widespread popularity as a basis for knowledge representation. The reason for this popularity is based on the fact that the knowledge represented by frames has a consistent structure, in particular for representing large volumes of data.

Many of today's AI tools and commercial knowledge based systems like KEE (IntelliCorp 1986), Keats (Motta 1986), KL-ONE (Brachman 1978b), Socerat (Socrates 1987), LOOPS (Bobrow 1983), SmallTalk (Goldberg 1984), KRL (Bobrow and Winograd 1985), FRL (Roberts and Goldstein 1977), ART (Laurent 1988) and many others, have utilised frames and their characteristics as the basis of a knowledge representation scheme.

A frame is basically a data structure for holding various types of knowledge. Conceptually, a frame represents an item i.e., a physical object or a concept i.e., an idea. The contents of a frame then describe that item by its characteristics, its properties or its behaviours.

The internal structure of a frame consists of a set of individually named slots in which the knowledge associated with that frame is stored. Values or facts that are related to

the frame are the main information stored in slots. There are many ways of storing these data; some values are stored as numbers while others might be stored as symbols, character strings, graphical or pictorial data. The content of a slot may be a single value or a set of values.

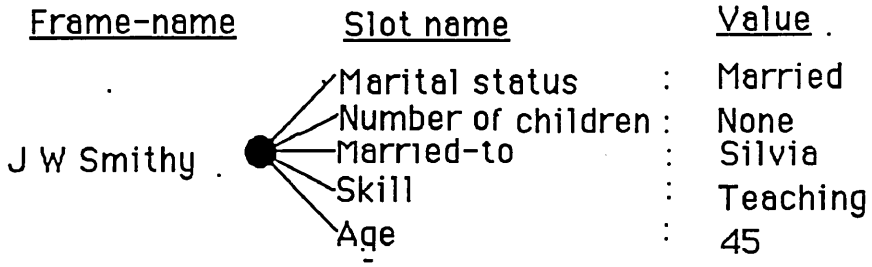


Figure 2.3, an example of a frame.

In Figure 2.3, an example of a frame is given. In this example, J W Smithy is the name of the frame. The frame contains five slots each with a single value. If J W Smithy had more than one skill, the Skill slot would have been multivalued.
In order to provide consistency and to facilitate better reasoning and inheritance, some restrictions may be applied to slots contents. There may be a restriction on the form of representation i.e., that the value of Age slot above, has to be an integer. There may be restriction on the values of a slot i.e., $0 \leq$ age ≤ 150, or restrictions that are independent of specific values i.e., maximum number of values for Married-to slot is only one or the value that can be chosen for Marital status is either Married or Unmarried. These restrictions are used to ensure that the values have proper form of representation and are interpreted correctly.

Restrictions can be imposed on the way that properties are inherited from higher level frames to the lower frames. These restrictions varies from one implementation to another. In KEE for example, there are twelve constraints on how slots and their values can be inherited (IntelliCorp 1986). These types of restrictions may be embedded in the frame structure or can be inherited from higher levels of the hierarchy. Inheritance restrictions may include union, intersection and override of the inheriting slots/values.

In figure 2.3, a frame is used to represent certain characteristics of J W Smithy. These characteristics were shown by several slots. In order to implement various restrictions on these slots, and their values and inheritance within the J W Smithy frame, a structure called a facet is introduced. A facet is a mechanism which contains all the information relevant to a slot. A variety of facets may be attached to a slot, each providing a different type of control parameter or characteristic. To perform any particular type of action in connection with a slot value, the contents of the appropriate facet are checked. These actions may include fetch, store, display or query.

Unlike semantic networks, procedural knowledge, in addition to declarative knowledge, may be attached to frame slots. Such procedures may be invoked when a certain condition is met. In J W Smithy frame, a procedure can be added to the frame such that, when the Married-to slot is filled with the value J W Smithy's wife name, it will be invoked and its effect may be to instantiate another frame representing the wife.

Brachman suggested that there are two main types of procedural attachment :

1) Meta-descriptions.
2) Interpretive intervention (Brachman 1977a).

The first type is expressed as knowledge within the knowledge base (probably as an individual concept). The second type consists of instructions to the knowledge base interpreter in the same form as the interpreter itself (written in the same language in which the interpreter is implemented). Winograd in his paper, explains how and when procedures may be attached to a frame (Winograd 1985).
It was mentioned that, there are three types of knowledge that can be stored in a slot; basic facts or values, constraints on slot values and inheritance, and procedural knowledge. However, additional types of information can be placed in a slot. For example, another frame or a pointer to another frame, or rules or an entire rule set, might be placed in a slot, permitting rule based knowledge as well as declarative and procedural knowledge to be structured in the hierarchy.

Despite their diversity, the concepts represented by frames often do bear some relationship to each other. This is part of the power of the frame-based representation which is the ability to capture such relationships in the knowledge base.

In a frame-based network, as with the nodes in a semantic network, frames can be used to represent either generic or individual concepts. A generic concept represents both general attributes and default properties. Individual concepts can inherit those general properties. As with a semantic network, the inheritance of properties is subject to exceptions which are imposed by inheritance constraints.

Another notion suggested by Winston, was to embody in the frames system, what he called "view changing" (Winston 1975). This has been interpreted as defining the knowledge base within different contexts. In other words, the interpretation of the knowledge can be different under different circumstances. In KEE and ART, "world" and "viewpoints" are examples of Winston's suggestion (Laurent 1988).

2.31 FRAME-BASED INFERENCE

The knowledge stored in slots of a frame can be available to inference mechanisms. For example, a rule can reason about the characteristics of a frame by referring to its slot values. In J W Smithy frame, there is an Age slot which has a value of 45. A rule such as the following may be embedded in the knowledge base :

Rule n : If Age ≥ 65
Then "Frame-name" is to retire.

If the age of J W Smithy is more than or equal to 65, he should be retired.

As mentioned above, in a frame-based network, knowledge is organised within each hierarchy with generic frames at higher levels and individual frames at lower levels. This provides the system with further reasoning ability: hierarchical reasoning. It can be inferred for example, that J W Smithy with all his characteristics, is a Male, is a Human and is a Mammal.

Frost has suggested five different types of reasoning method that can be implemented in a frame-based system (Frost 1986) inferred existence, inferred generic properties, default properties, recognition of abnormal situation and inference by analogy.

By inferred existence, the inference mechanism can deduce the existence of a given entity in the knowledge base. In the second method inferred generic properties, the system can infer that the entity in question has all the generic properties. The third method is value inheritance by default for a particular slot of a given entity. In the fourth method, the absence of a value for a slot of a given concept can be interpreted as an unusual situation. This may cause contradiction since in various cases the query frame may have incomplete information. In the fifth method of inference, analogy is made between the entity in question and a frame in the knowledge base, and as a result of this analogy the missing part of the given query frame can be completed.

2.4 KNOWLEDGE REPRESENTATION LANGUAGES

During the second half of the 1970's, knowledge representation languages emerged as a separate vehicle for representing knowledge. Several different languages developed, such as KL-ONE (Brachman 1977a), KRL (Bobrow and Winograd 1985), FRL (Roberts and Goldstein 1977). More recently, powerful hybrid toolkits for building commercial knowledge based systems were developed including KEE (IntelliCorp 1986), ART (Laurent 1988) and LOOPS (Bobrow 1983). These systems embody hybrid programming environments which combine procedures, frames and object-

orientation and rule-based paradigms in one language. In the following subsections, detailed discussions on KL-ONE and KEE are presented.

2.41 KL-ONE

KL-ONE is a well known frame based knowledge representation language and is based on SI-Nets (Structured Inheritance Networks) which were introduced by Brachman (Brachman 1977a, 1977b, 1978a, 1978b). In SI-nets, the emphasis is on epistemological issues (the internal relationships that a concept has with its properties and their constraints). Since it was introduced (Brachman 1985b), KL-ONE has been used in a number of applications, ranging from natural language understanding, to question answering systems, to the modelling of office automation. KL-ONE is more than just a representation language, as it includes facilities for the building, storing and interrogation of the network. It has been evolving and updated with new ideas since its creation. There are many offshoots from KL-ONE that have been developed under its strong influence. NIKL (New implementation of KL-ONE) by Kaczmarek (Kaczmarek 1986) and KRYPTON (Brachman 1983a) are new implementations of KL-ONE.

KL-ONE contains three general types of objects:

1) Concepts.
2) Roles.
3) Structural-descriptions (SDs).

Concepts are regarded as the basic elements of KL-ONE and are defined as formal objects employed to represent objects, attributes and relationships of a particular domain. There are three types of concepts; generic, individual and parametric individual (paraindividual). A generic concept represents a class of individuals (description of the template-like member of the class). An individual concept represents a specific object, relation, etc which matches the generic concept's description (what Brachman calls "individuation" of a generic concept). The third type of concept, the paraindividual concept (PIC), defines relationships between two or more of the concept's roles. Role/filler descriptions and structured descriptions (SD's) in KL-ONE, were developed to address both the internal structure of a concept and the relationships with other concepts. Roles represent the conceptual sub-structures of an entity, such as its attributes or its parts. An example of the use of a part sub-structure would be the fingers on a hand, whilst an example of an attribute sub-structure could be the colour of an object or the arguments of a function, such as

the multiplier and multiplicand in the multiplication operation. There are two types of roles:

1) Generic roles of a generic concept (called RoleD) describing generalised attributes of that concept.
2) Instance roles, representing the relationship of a particular individual concept with either a generic role (called RoleF), or an individual concept. Unlike concept names, roles may have the same names even if they are part of the same concept.

A generic role's attribute description is provided by the role's facets, which are as follows :

1) the V/R (value restriction) facet specifies a generic concept, which is a description that any filler must satisfy.
2) the Number facet indicates the number of fillers of the particular role to be expected. It may be a pair of numbers specifying the range.
3) The modality facet controls the action of individuation and specifying the importance of the attribute to the concept.

In general, roles represent properties of a concept and a collection of roles is regarded as a formal entity that relates the functional role, the content in which that role is played, and a set of fillers of the role. In other words, roles are not only used to specify numbers of fillers but also to specify how those fillers may be used within the conceptual structure.

While a role indicates that for any instance of the concept there will be appropriate numbers of fillers for the given functional role, an SD (Structural Description) indicates that any instance of the concept will represent relationships specified in that SD.

A set of SDs are used in the concept to specify certain constraints on how the concept's role fillers may interact with each other, to partly define the concept itself. Each SD is a set of relationships between two or more concept roles, these relationships being expressed by paraindividual concepts (PIC).

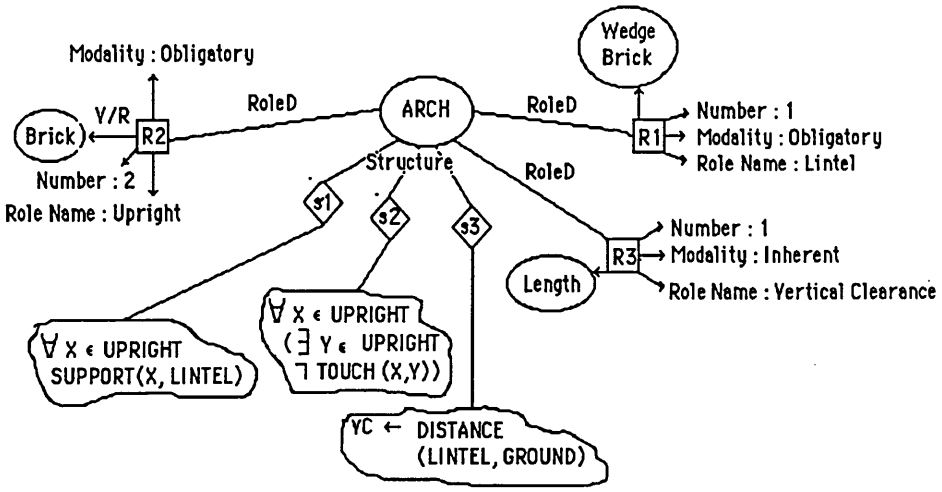


Figure 2.4, a KL-ONE concept for a simple arch.

In figure 2.4, a simple concept representing an Arch with all its internal substructures is shown. There are three roles R1, R2 and R3, accompanied by three structural descriptions (SDs). R1 is linked to concept Arch by Role-D link and specifies that this particular Arch has one 'LINTEL' which must be a WEDGE-BRICK. R1 has four facets:

1) the link name modality which specifies the level of importance of the attribute to the concept.
2) V/R (value restriction) link specifies the role filler.
3) the role name link, specifies the name of the relationship between the filler and the concept.
4) the Number link specifies the number of fillers in the role.

R2 specifies that the concept Arch has two UPRIGHTS of type BRICK. R3 specifies the VERTICAL CLEARANCE. The three SDs, are used to specify the relationships between the roles facets. S1 for example, indicates that how every UPRIGHT supports a LINTEL.

2.411 INHERITANCE IN KL-ONE

Inheritance in KL-ONE is confined by the relation that connects two formal objects of the same type, concept to concept, role to role and SD to SD.

As mentioned above, the relationship between two concepts is called individuation, so that there is always a concept that is being individuated. The individuator must satisfy all sub-descriptions of the individuatee. This implies that, not only is there a relation between concept and sub- concept but there is also a set of sub-relations between the

concept roles. In other words there is a set of sub-relations between the generalised attributes of that concept and the values of those attributes in the individuator.

The notion of inheritance in KL-ONE may be illustrated by saying that sub-concepts themselves may be generic and can be formed by restricting the inherited characteristics from the concepts.

The structure of a taxonomic hierarchy is merely based on the formation of more and more specific descriptions. In such structures, there has been, in general, a single link, for example an IS-A link, to specify inheritance along the hierarchical chains. The assumption is that everything relevant to the higher classes is relevant to the lower classes as well.

In KL-ONE, the roles and SD's of a parent concept will each contribute to the inheritance process between the lower level concepts and the parent concept. Brachman regards the inheritance link as a cable carrying down roles, and SD's as a group, since their descriptions are entirely dependent on the parent concept. These inherited properties must be controlled under strict conditions by controlling their modification according to specifications held in the lower level concepts. These modifications for each role and SD are specified and monitored by inter role or inter SD link, between the original role and the new one.

There are currently three types of role modification in KL-ONE; the first, satisfaction which is the process of filling; the second, differentiation which is creation of sub-roles; and the third, the restriction of the role itself.

2.412 PROCEDURAL ATTACHMENT IN KL-ONE

There are two types of procedural attachment (Brachman 1978b) :

1) meta-description, which is meta-knowledge about the actual knowledge, and has the same form as the knowledge structure.
2) interpretive-intervention which expresses direct instructions to the interpreter in the language that implements the interpreter itself.

In the case of meta-description, the interpreter is being instructed to make a type or level jump when processing a concept. In KL-ONE meta-information is information about a formal entity, which can be a concept or a role or an SD, and is directly (ie. explicitly) linked to an appropriate node. This link is called a metahook, and can be attached to a concept, a role or an SD. There is another type of link in KL-ONE called

interpretive-hook (I-hook), which is used for attaching interpreter code directly to a concept role or SD. An I-hook points to an entity in which direct instructions to the interpreter are expressed in the same language as that in which the interpreter is implemented.

FRL was one of the first frame-based knowledge representation languages that was developed (Roberts 1977). KRL was introduced around the same time. KRL is also one of the first Frame-based knowledge representation languages, and attempts to integrate procedural knowledge with declarative knowledge (Winograd 1985, Bobrow 1985). In KRL (Knowledge Representation Language) the formalism for declarative knowledge is based on "structured conceptual objects" with associated descriptions. These objects form a network of memory units with several different types of links, each having well specified implementations for retrieval process. Procedures can be associated with the internal structure of a conceptual object, to allow the steps for a particular operation to be determined by the characteristics of the specific entities involved.

The above frame-based knowledge representation languages were developed in late 70's, and their influence on recently developed knowledge representation languages are apparent. KEE, LOOPS and ART for example are hybrid environments that facilitated the development of frame based applications in addition to utilising rules and procedural representation.

2.42 KEE

KEE (Knowledge Engineering Environment) is one of the most powerful knowledge based systems available. This system was introduced to the commercial market in 1983 (IntelliCorp 1985, Laurent 1988), and at present is available on most of the lisp-machines, Dec-Vax workstations and Sun workstations. The KEE system is a hybrid system consisting of Frames, certain aspects of Object-Oriented Programming techniques, rules; and has the ability to access the implementation language, Lisp. It provides a well designed representation language based on 'units', a data structure similar to frames. These units represent prototype descriptions of objects. Each description in KEE consists of five components :

1) Units - which represent objects
2) Slots - define attributes of units
3) Slot Values - are the values of attributes
4) Facets - describe slots
5) Facet Values.

Facets are used to attach several types of information to a slot in addition to its values. This information can be a procedural attachment, inheritance specification and/or meta-knowledge.

2.421 GENERAL FORMAT OF UNITS IN KEE

Each unit is defined by its unit name and a set of properties represented by slots. There are two types of slots; owner-slots and member-slots. Member slots are used to describe class members' properties and may be inherited by instances of the class unit. Owner slots are merely there to define the unit it contains. Slots have a number of facets in addition to the value facet; these facets are inheritance roles, the value class, maximum and minimum cardinality, and comments.

The inheritance role facet specifies how a slot will inherit attributes from its ancestor's slots. The facet Value class specifies types of values. The cardinality-maximum-and-minimum facet defines a range in which the number of values are specified and the last facet of a slot is a set of values belonging to that particular slot. Some slots have comment facets which may be used for interaction with the outside world.

The value class specification, and the limitation imposed by cardinality, are particularly important since they grant the frame language the ability to represent quantified assertions and to make appropriate inferences. It may be reasonable to define the type of hierarchy used in KEE as a hierarchy of descriptions, ie. a system based on classification (see figure 2.5).

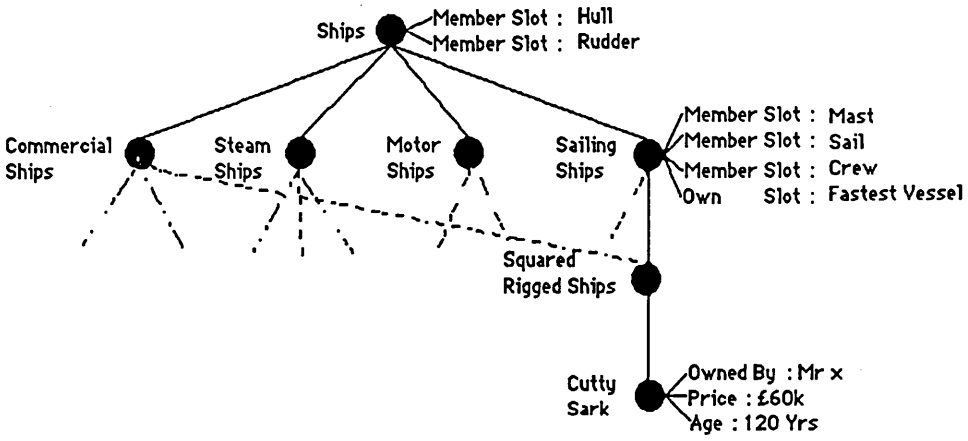


Figure 2.5, Shows a classified hierarchy.

Each frame represents a class of objects and a link (not specified but it could be IS-A link) connects a class to its super class.

In Figure 2.5, a classified hierarchy is shown. Ships is the superclass object, and its immediate children represent the class objects in the hierarchy. Square Rigged Ships is an example of class instance, and Cutty Sark is an individual object. Note that in this example, the object Square Rigged Ships has two parents. In KEE, the concept of multiparentage has been considered whereas in LOOPS, all the hierarchies are trees and there is no multiparentage (Bobrow 1983).

2.422 INHERITANCE IN KEE

The mechanism of inheritance in KEE is provided by passing down relevant information from one level of hierarchy to its lower level, and it is controlled by various restrictions specified in the facets of each slot. These restrictions include inheritance role, value restrictions, and relevant procedural restrictions (if any).

KEE supports multiparentage, ie a child unit inherits properties from more than one parent unit. In figure 2.5 for example, Square Rigged Ships may inherit properties from both the units Commercial Ships and Sailing Ships. There are 12 possible inheritance roles within the KEE system, as follows :

1) Override values : this is a default value, so that if no particular inheritance role has been specified, then the inheritance from the parents is overridden by the child's value, if the child has any explicit values.
2) Union : this inheritance role takes the union of the values which are inherited from the parent's nodes and the actual unit itself. Thus, it combines the values of the child's slots with the values of the corresponding slots of any parents, without producing duplicates.
3) Runion : this is the reverse of union.
4) Save-values : inheritance of this form requires that each parent value list be equal to the child's explicit value list.
5) Unique-values : unique inheritance blocks inheritance of slot values altogether. The explicit values of the slot are retained as the derived values.
6) Variable-values : Variable-values inheritance is even stronger than unique values. It not only blocks all inheritance, but also suppresses the KEE system's usual policy of noticing when a slot has been modified.
7) Maximum : inheritance of this form chooses the maximum of the local values and parent values.
8) Minimum : inheritance of this form chooses the minimum of the local values and the parents values.

9) Method : inheritance of this form works by decomposing each explicit value into main code, before code, after code, and wrapper code parts (methods are described in later stages).
10) Vcsimplify : inheritance of this form enables value classes to combine and simplify inherited specification.
11) Union-Each-Value : inheritance of this form expects the individual values in the slot to be lists and then it performs a union on the corresponding positions in each slot's lists of values.
12) Runion-Each-Value : this is Union-Each-Value in reverse.

Value class is another facet of a slot. It provides partial or full specification of values that a slot can have. Value class can be specified in different ways. Not-one-of, for example, indicates an explicit set of values which are not allowed. Thus, value class is another type of constraint on the inheritance mechanism.

Another facility[3] offered by the KEE system is called KEE worlds. This is an environment with more than one context. That is, a slot in a frame is described as containing sets of values, each set pertaining to a different universe, or, to the same universe at different points in time. Under real-time conditions, there are many types of problems that require knowledge represented in different context. For example, consider the effect of weather on a journey taken by an aeroplane, using the same route, in one year.

Multiple context is one of the important facilities that most of todays Hybrid knowledge based systems offer. The developers of ART call it "viewpoint" (Chung 1988), where as KnowledgeCraft relies on a truth maintenance mechanism embedded in its OPS5 forward-chaining mechanism (Laurent 1988).

Reasoning in the KEE system is based on rules, which are represented as units, like any other KEE object, and can be used in either forward or backward chaining. In KEE a reasoning process is usually initialised by a call to the knowledge-base Assert and Query Language, supplied in the package. This language, called TellAndAsk, provides three different functions:

1) assert; to create a fact,
2) retract; to remove the fact,
3) query; to extract knowledge from the knowledge-base.

---

3 This facility has been developed under the influence of Winston's idea about "view changing" (Winston 1975).

2.423 PROCEDURAL ATTACHMENT IN KEE

In the KEE system a knowledge base may be a combination of descriptive knowledge and behavioural knowledge. The behavioural knowledge (or procedural knowledge) is represented by methods (functions or procedures) . Each method is a program, and whenever it is called or triggered, the program is run and the appropriate task is performed. Methods can be inherited through the hierarchical levels of the knowledge base. Further, methods can be activated indirectly, as a result of the consequences of another method or an active value. An active value provides the facility to monitor what is happening to a particular slot; it represents the data directed programming aspect of the KEE system. Active values are represented in the knowledge base as units and can be attached to slots in units, so that they fire when a triggering event happens. It seems that active values represent the notion of meta-description and, methods represent interpretive-intervention, as described in KL-ONE.

There are other hybrid knowledge representation languages available in the market; LOOPS (Bobrow 1983), KEATS from Open University (Motta 1986), ART (Automatic Reasoning Tools) from Inference Corporation (Laurent 1988) and KC (Knowledge Craft) from Carnegie group (Chung 1988). All these tools use frames as their basic block for knowledge representation, and inheritance of properties and rules are used as the main inference mechanism.

2.5 PRODUCTION SYSTEMS

The Production System is one of the oldest and most commonly used models for knowledge representation and its associated application[4]. The production system model has been used successfully to solve a wide variety of problems, such as medical diagnoses (Shortliffe 1976), and automatic configuration of computers (McDermott 1982). Production systems were originally developed by cognitive scientists to model human memory and problem solving (Simon 1965, Newell 1972, Newell 1976, Davis 1980).

Many of the application systems created with the production system model are of a class known as "expert systems". In AI, the term expert system is used to refer to a computer program that is able to perform within a specific domain at the level of a human expert in that domain.

---

[4] It started with Post using production systems in symbolic logic (Post 1943), with the Markov algorithm in mathematics (Markov 1954), rewrite rules in linguistics (Chomsky 1957). Simon and Newell used the production system in their Knowledge Based System for chess analysis (Simon 1965) and later for human problem solving (Newell 1972).

Note that, while production systems are often referred to as rule based systems, the term "rule based" actually has a slightly broader definition. For example, logic programming is also a type of rule based programming (Brownston 1985).

However, the computations in production systems are different in style from computations performed by procedural languages (such as Fortran), functional languages (such as pure-Lisp or Hope), or object oriented languages (such as SmallTalk). One of the main differences is the production system's use of "data sensitive" unordered rules, rather than sequenced instructions as the basic unit of computation.

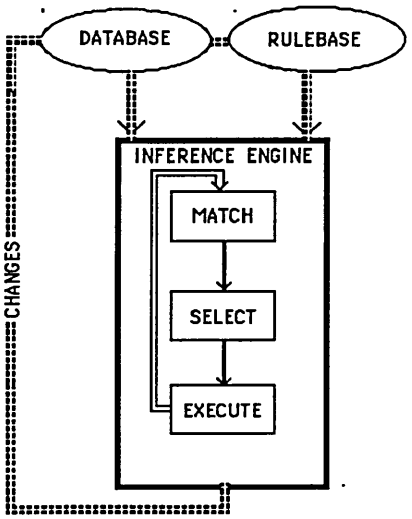


Figure 2.6, Computational model of production system.

Figure 2.6 shows the basic architecture of a production system computational model. A production system contains a set of conditional statements known as productions (production rules, or simply, rules), a collection of given or derived facts known as the database (data memory or working memory), and an inference engine which implements the invocation of the rules as a sequence of "modus ponens" actions (Hayes-Roth 1982).

The inference engine operates in cycles where each cycle consists of three phases; match-rules, select-rules and execute-rules. In the first phase, match-rules, the inference engine finds all the rules that are satisfied by the current contents of the database according to the matching algorithm. The match may involve the condition part, the action part, or both parts of the rule. All the matched rules (also called conflict set) are potential candidates for execution. The same rule can appear in the conflict set several times if it is satisfied by different sets of data items. In the second phase, select-rules, the inference engine applies a control strategy (conflict resolution) to select the appropriate rule which is to be executed in the third phase; execute-rule. Executing rules is also referred to as "firing" rules, by analogy with the

firing of neurons. In the next cycle, the same process will start over again. This repeated action is referred to as the "recognise/act cycle". The part of inference engine that fires the rules by interpreting the conditions and actions of the rules is called the "rule interpreter". Note that, the term interpreter is often used to refer to the entire inference engine.

2.51 DATABASE IN PRODUCTION SYSTEMS

In the simplest production system, the database is simply the collection of symbols intended to reflect the "state of the world". For those systems intended to explore the symbol processing aspects of human cognition (Anderson 1978), the database is interpreted as modelling the contents of a certain memory mechanism (ie. short term memory). For systems intended to be used for consulting or advising experts, the database contains "facts and assertions" about the world (Forsyth 1985).

Most databases are global stores; they hold knowledge that is accessible to the entire system. The items in the database are referred to as elements, and the representation of elements varies from strings to complex structured objects. In KEE for example, frames are used to represent objects (IntelliCorp 1986); in ROSIE, data and assertions are stored in a relational database of n-ary relations (Schank 1972); and in OPS5, elements in the database are in the attribute-value format (Brownston 1985).

In addition to the problem solving strategy embodied in the inference engine, the state of the database is another control over the execution of the production system program. It is the contents of the database that determine which portion of the production system program is available for execution. That is, at any particular time, only those rules that match the data in the database are executable, and the inference engine determines which rule is actually to be fired.

The database provides another control over the execution, by being the only means of communication between rules. That is, there is no mechanism for passing data from one rule to another directly, as one would do with a parameter in a procedure call. The elements in the database may be created, modified, or removed.

2.52 RULES

The left hand side of the rule is the condition part, also termed the antecedent or situation. The right hand side is the actions part of the rule, also termed the consequent. The left hand side of a rule is usually a Boolean combination of clauses. The

type of Boolean operators vary with the production system language. Many languages allow only the operators AND and NOT. A clause (sometimes referred to as a predicate) specifies a restriction on the value of some particular property or attribute of some object that can be represented in the database. The right hand side, or action part of the rule, is generally a list of modifications to be made to database when the rule fires. These actions usually add, modify, or delete elements in the database, but they may also perform external communications with the outside world.

Interaction between rules is very limited; they can communicate with one another only by way of data in the database. However, it is possible to provide interaction by having a very large short term memory (STM), and a complex message passing algorithm (Davis 1985).

2.53 INFERENCE ENGINE

The inference engine is the source of much of the variation found amongst different systems, but it may be generally defined as a "select/execute" loop (see figure 2.6). In each loop, a selection mechanism is repeatedly applied in order to choose a rule applicable to the current state of the database, and then that rule is executed. Sometimes the action of inference engine results in a modified database, and the select phase begins again.

Rules in a production system can be applied in either direction which corresponds to the type of reasoning strategy employed by the inference engine. The inference engine looks in the database to see which rules have their situation parts satisfied, selects one of them and fires it by performing the corresponding action. The rules that get fired are thus chosen by information in the database; such systems are called "forward chained" (or data-driven). The generalised rule "S => a" can be interpreted as : " if condition S holds, a is a consequence", but the same rule is subject to a slightly different interpretation : " if a is to be established, try to establish S".

Looking at rules in this second way suggests an alternative execution strategy. At each cycle those rules whose right-hand sides were relevant to one or more of the current goals would be located. If the situation of such a rule was satisfied by the database then using that rule would be a step towards satisfying the goals; if not, the next cycle begins with a new set of goals.

This alternative procedure clearly uses the same information, but the selection of the rules to be fired is determined now not so much by what is in the database, but rather by what is expected or hoped to be found. This method is referred to as "backward

chaining" or a goal-driven strategy. Both methods have their respective advantages and disadvantages. Often, the major factor in making a selection arises from the particular domain in which the knowledge is applicable (Jackson 1990). Strategies which combine the merits of both approaches have been suggested (Jonson 1988).

2.531 MATCHING RULES AGAINST DATA

There are many types of matching algorithm that must be accommodated in a production system (Sell 1983) :

a) A simple identity known as "literal match".
b) A matching based on patterns.
c) A matching based on unification, which supports variables on both sides.

For example, suppose two of the rules are :

If it is raining, then the ground is wet

If height of X > height of Y, then X is taller than Y

where X and Y are variables, and the database contains the following items:

It is raining
The ground is dry
Height of Jon = 6
Height of Ian = 5
Jon is taller than Y

A literal match will satisfy the first rule since "it is raining" in the database matches exactly the condition part of the first rule. Pattern matching can satisfy the second rule by letting X take the value Jon and Y the value Ian. The last item in the database would allow the system to fulfil the condition of the second rule by unification, letting X take the value Jon. (The meaning of the last item is that Jon is taller than anyone else in the database).

The simplest action to be performed when the left hand side of a rule is satisfied, is that of replacement, where an old item in the database is replaced with the new one. In the example above "the ground is dry" would be replaced by "the ground is wet".

The next level up is addition, which aggregates items in the database. This is in fact the most frequently used action. In the example above, the system could add "Jon is taller than Ian" to the database, and if it used unification, it could add "height of Jon > height of Y".

2.532 UNCERTAINTIES

Irrespective of whether the inferencing procedure works backwards or forwards, it will usually have to deal with "uncertain data". Real-life problem-solving is usually so complex, and the available facts so incomplete, that uncertainty must be accounted for in order to produce useful answers. Uncertainty can be dealt with by a number of different methods :

a) Fuzzy logic: this allows the representation of partial truth within Boolean logic. A "1" is taken to represent truth and "0" to represent falsity; the real numbers between these values then indicate all possible shades of partial truth (Zadeh 1974).
b) Bayesian logic: this scheme is based on probability theory. Bayes' rule provides computation of relative likelihoods between competing hypotheses on the strength of the evidence (Duda 1981).
c) Certainty factors: Shortliffe (Shortliffe 1976) devised a scheme based on what he called certainty factors for measuring the confidence that could be placed in any given conclusion as a result of the evidence so far. As new evidence is established, confidence estimates can be revised.

Many other schemes dealing with uncertainty do exist. However, the methods mentioned above have been used in a large number of expert systems, and do appear to work satisfactorily (Michie 1984).

2.533 CONFLICT RESOLUTION

The performance of the inference engine and the production system as a whole, depends on the conflict resolution strategy for both sensitivity and stability (McDermott 1984). Sensitivity is the system's quickness of response to the dynamically changing demands of its environment; while stability is the system's continuity of behaviour.

In the matching phase, more than one rule can be triggered (selected or instantiated). The way in which a rule is selected from a set of triggered rules is called the "conflict resolution strategy". There are various methods of implementing such strategy :

a) Refraction : In this method it is required that rules fire not more than once on the same data. This is intended to prevent a form of infinitive looping that could occur if a rule did not change the contents of working memory.
b) Data ordering : This is a powerful way of adding sensitivity to a conflict resolution strategy. In this method, the data is ordered by recency or activation. A recency or activation ordering gives preferences to rules that

have matched elements to database most recently, or that are strongly related to recently added data.

c) Specificity ordering : Specificity favours rules that are special cases of other rules, or are more specific according to some measure.

d) Rule ordering : This strategy provides a static ordering of the rule set independently of the way the rule is instantiated by data. Using static ordering tends to be less sensitive, since priorities are independent of the instantiation of the rules.

Note that none of the principles described above guarantees that only a single triggered (instantiated) rule will remain in the conflict set. If a single firing is required on each cycle, an arbitrary decision can be made so that a single rule can be selected. The alternative to arbitrary selection is the firing of all the triggered rules (Rosenbloom 1984).

2.54 PRODUCTION SYSTEMS CHARACTERISTICS

A production system offers several important characteristics which yield certain advantages and disadvantages; knowledge is separated from the control mechanism, it has a restricted format, and there is a limited interaction between rules.

A consequence of the separation of knowledge and control is that production systems can cope with unanticipated situations. In other words, unplanned interactions result from applying knowledge when it is appropriated rather than calling on it in predetermined sequences (reactivity). Because knowledge is stored in separate units, rules can be modified with very few side effects (modifiability) and rules are relatively easy to explain (explainability).

As a consequence of the restricted format all facts are stored in a similar form and only a simple inference engine is needed (simplicity of control). Restricted format leads to machine readability; rules are machine readable, and a limited amount of automated modification and explanation, consistency checking and learning, is made possible.

As a result of the limited interaction between rules and the inference engine, rules tend to be modular (modularity). This facilitates reactivity in the system and the explanation and modification of rules. Rules easily express basic symbol processing acts (expressibility).

There are also some disadvantages with production systems. Even for a simple task, the stepwise behaviour of a production system is rather unclear to the user. Reevaluation

of the database, and scanning the entire rule set, in any cycle, contributes to this lack of clarity (opacity). Because the matchers in production systems have to reevaluate the whole situation to find applicable rules on each cycle, most production systems run much slower than procedural programs.

2.55 THE EFFICIENCY OF PRODUCTION SYSTEMS

Efficiency is an important consideration in production systems, since expert systems or any other application of production systems, may be expected to exhibit high performance in interactive real time domains. Thus, for a production system to be efficient, all aspects of the system should be considered. Although correctness, readability of code, clarity of representation and good documentation are equally important issues, only the execution of a production system is considered here.

As discussed above, the execution of the production system operates in cycles. Each cycle consists of three phases; matching, conflict resolution and action. In the matching phase, the productions are examined by the interpreter to see which are appropriate and could fire. If more than one is found to be appropriate (triggered), a strategy will be applied to choose one or more rules from conflict set. Finally, in the third phase the selected production(s) will be fired.

However, it has been suggested that the matching phase takes up 90% of the computational resources and time (Stalfo 1985), and as production systems have become larger and more complex, the point of efficiency has necessitated the construction of a more complex data structure in both the rule base and database. Some production systems with a large rule base, for example, have employed partitioning or indexing mechanisms, rather than scanning through all the rules.

In the following subsections, these issues may be better illustrated by brief descriptions of two familiar mechanisms used in production systems; the Rete algorithm, and the blackboard mechanism.

2.56 THE RETE ALGORITHM

The Rete algorithm is one example of the many different techniques used to reduce the time spent on the matching phase (Forgy 1982). This algorithm is employed in the OPS5 production system language, which employs complex pattern matching (Brownstone 1985). A Rete network is constructed from the left hand sides of productions, to represent a network consisting of nodes (Gupta 1985). Each node represents an abstract operation to be performed during the match phase, and is

interpreted by the inference engine at run time. In the matching phase, the objects that are passed between nodes in the network are called tokens. Each token consists of a pointer to a list of elements in the database, that matches the condition elements of the left hand side of a production.

The Rete algorithm takes into account the fact that only a small portion of the database is changed at each cycle, and exploits the stored results of previous match cycles, eg. where there are similarities between condition elements (RHS) of productions.

It is important to notice that the Rete network does not reduce the modularity of rules, but adds a certain type of locality by partitioning the rules into relevant groups.

2.57 BLACKBOARD DATA STRUCTURE

Production systems have improved in efficiency, and now have the capability to represent more complex situations. The Hearsay system exemplifies such an improvement (Barr 1981, Nii 1986). Hearsay is a speech understanding system, which is regarded as an important development in AI with respect to its functionality and construction. The area of application was the understanding of spoken requests for literature in a computer science database (Erman 1980).

The structure of Hearsay is based on the assumption that the knowledge of several experts are represented by knowledge sources. To solve a given problem, each independent knowledge source writes a suggestion on a global blackboard data structure where every other experts can see it. This is the way that experts (knowledge sources) communicate with each other. The entry on the blackboard triggers the other experts on which they will take the analysis further. This process continues until the system arrives at a conclusion.

The Hearsay system consists of a number of knowledge sources, and a backboard containing different levels of knowledge. The knowledge sources operate in between and in different levels of the blackboard. The blackboard is a global database in which the hypotheses and the supporting criteria can be stored. During the evolution of the Hearsay system, the number of knowledge sources was increased from three to twelve. In Hearsay-2, the latest version, there are twelve knowledge sources, each of which is devised to deal with certain levels of the hierarchical segmentation of speech.

The knowledge sources in Hearsay are rule-based, and the reasoning mechanism utilises both forward and backward chaining. The Hearsay control mechanism can switch between these two reasoning mechanisms by checking the direction of the given

credibility rate of each competing hypotheses. The knowledge sources that start from lower levels to higher levels, are data driven (or forward chained) and those knowledge sources that take the direction from higher levels to the lower levels, are goal driven or backward chained.

The construction of the Hearsay system is important, due to its modular architecture of knowledge sources. There are no direct relationships between the knowledge sources. The only means of communication between knowledge sources, is the blackboard. This provided great flexibility during the evolution of the system, when different combinations of knowledge sources and control strategies were tried (Nii 1986). Many applications have used this blackboard architecture including HASP/SIAP, CRYSALIS, TRICERO and ACAP (Alty 1988).

2.6 CONCLUDING REMARKS

One of the aims of the project was to select an appropriate model of knowledge representation, that could be used as the basis for developing a parallel language, suitable for the architecture of the SM. The SM is the name given to a simulation of a parallel architecture, which was developed prior to the start of this project (Loh 82a, 82b). It consists of a rectangular array of Processing Elements ("PEs"), where each PE is connected to its nearest neighbour (see chapter 3). Although the architecture has been modified, it has nevertheless, imposed certain constraints on the selection of an appropriate knowledge representation model.

This architectural constraint, and the primary decision of not employing mathematically based representations, has reduced the range of possible options. From these options, only associative networks; semantic and frame-based networks and, production systems seem to be the most suitable choices.

Neural nets share the concept of associativity with semantic and frame-based networks, but the major difference that they have with these nets is with their hardware characteristics. In neural nets, unlike the SM, only simple PEs with rich inter-connections are employed. Knowledge is then represented as patterns which are produced by those inter-connections (Beale 1990). In chapter 3, neural nets are discussed in more detail.

Semantic networks and frame-based networks share the important characteristic of an association between every two objects of a particular domain. This association not only represents the relationship between the two objects, but also provides a direct path from one object (or concept) to its most relevant associate. These paths can later be

utilised as directions for propagation, network traversal and communication between relevant objects in the network.

In production systems there are no direct relations between rules. In each cycle to find the next appropriate rule, the system has to scan through all the rules. There are indexing or partitioning algorithms that can be added to the system to reduce the search space, like the Rete algorithm. Nevertheless, these mechanisms can not provide the desired explicit relationships, that are freely embedded in associative networks. The local distribution of knowledge reduces the search space and the amount of communication. Further, accessing and utilising a separate component (ie index table), will reduce the amount of parallelism and increase the communication overheads (Krishnamurthy 1989).

In the SM, with a distributed parallel architecture, one to one mapping of each node of the associative networks to a PE and the utilisation of embedded relationships amongst the nodes, is far more feasible than the implementation of a production system.

Another important feature of associative networks is the inheritance of properties. In this mechanism, the general or most abstract properties of generic concepts can be inherited by their descendants at lower levels of the hierarchy. Therefore, "inheritance of properties" provides a distinguishable data economy within the system.

Production systems, however, offer robustness, consistency and modularity. In addition, production systems have been used extensively in industry, and consequently, there is a vast amount of empirical knowledge that may be useful in the construction of AI systems.

Implementing a parallel production system may seem feasible particularly with the modular characteristics of rules that it offers. But in a comparison of existing parallel production systems with associative networks, eg semantic/frame-based systems, one can see that it is very difficult to provide hard evidence of one having overall superiority. With the Connection Machine (Hillis 1985) and DADO2 (Stalfo 1987) for example, utilising semantic networks and production systems, respectively, as the knowledge representation formalism, there is no proven performance test which shows superiority of one formalism over the other.

The implementation of frame-based networks as the knowledge representation formalism, is probably the best choice in the SM. This is firstly because of the SM's distributed architecture which is well suited to the distributed knowledge in a frame-based system. Secondly, a frame-based network offers associativity, locality,

inheritance of properties, procedural attachments. Thirdly, each frame in the network can represent a complex concept with all its internal/external relationships. The study of today's AI tools (Laurent 1988, Chung 1988) highlights the flexibility of frame-based systems, where each frame can encapsulate different types of information and is regarded as a hybrid representation scheme. Such systems can embody different types of representation schemes including production systems, Object Oriented systems and procedural languages.

3.0 CHAPTER THREE : PARALLEL ARCHITECTURES FOR AI

3.1 INTRODUCTION

The computer industry has experienced four generations of development, which result from advancements in technology; from relays and vacuum tubes to VLSI (very large scale integration). A Cray computer for example, can now perform in the order of 100 million double precision multiplications per second (Hayward 1991). Today's complex applications require such high performance computers. Some of these applications include : modeling global weather patterns, analysis of the aerodynamic properties of a wing, The simulation of sub-atomic world of quantum theory (Tabak 1990).

Artificial Intelligence ("AI") applications also require high performance machines. An ideal AI machine must be able to explore large multi-domain knowledge bases, and make appropriate inferences to exhibit intelligence, or perform intelligent behaviour. A serial exploration of large volume of data will always be limited by the various time constraints imposed on the system, either externally or internally.

Basic theory in automata shows that, given enough time, a serial machine can, in principle, compute anything that a parallel machine can do (Nelson 1968, Shields 1987). As long as there is no time constraint for instance, it is certainly possible to ensure that the travelling sales man will arrive at his destination by checking all possible paths. Most computations involve searching to find the next appropriate move (eg in the game of chess), until the ideal situation is reached. In other words, in a search space that represents the problem, the purpose is to start from a node and try to get to the goal state. Each time that a move is taken, there would be a new series of options available to the searching mechanism. Gradually, the number of options become so great that it is almost impossible to investigate all of them (combinatorial explosion).

As a result of research in AI (Rich 1983, Schutzer 1987), cognitive science (Simon 1965, Uhr 1980), and other areas (Searle 1984, Appalaraju 1984), it seems that parallelism may be an alternative to increase the speed and the power of computation. In spite of optimised hardware/software, such as the development of fast Processing Elements ("PEs") (Classe 1990), or improved algorithms (Hillis 1986, Markov 1954), the problem's complexity and its sheer size necessitate parallel exploration of its search space. In order to achieve the ideal speed, it may be essential to exploit today's technology in a parallel environment. In particular, with recent developments in VLSI, WSI (wafer Scale integration) and other areas of computer science (Fox

1986), the possibility of direct implementation of parallel models into the hardware has greatly increased.

In the AI community, the theory of parallel computation is widely accepted; and one of the most influential elements in the development of parallel machines has been the human brain as a model for parallel processing (Albus 1981, Rumelhart 1987). Researchers like Fahlman (Fahlman 1981), Hillis (Hillis 1985), and the whole community involved with neural nets (Reilly 1984), for example, have been directly affected by their understanding of the brain's structure. There are other people whose understanding of the structure of the brain was based on the more cognitive, psychological aspects of the brain. Parallel production machines like DADO2 (Stolfo 1985, 1987), or machines whose structure is based on associative nets like semantic nets (Collins 1975, Bic 1984), are the results of such research.

In the human brain, with its large capacity and large number of relatively slow neurons (relative to today's circuit's speed), the speed in which knowledge can be retrieved is very fast (Feldman 1985). This may reflect the fact that knowledge is highly organised in the brain and is processed in a parallel environment. One interpretation is that the organisation of knowledge in our brain is based on semantic coding, which is the encoding of knowledge according to its meaning. In this way of encoding, knowledge has both explicit and implicit interpretations, and can be retrieved by traversing through patterns of activity, rather than just returning the output of a certain location in the conventional computer memory (Beal 1990).

The human brain is believed to have a structure that is in part parallel, and in part serial, and to contain in excess of 10^{12} neurons, each being connected to 1000-10,000 other neurons[1] (Uhr 1980). A neuron is the basic building block of the human brain, and propagates an electrical signal along its length and to other cells. Because of its size and structure, a neuron can sustain only a relatively slow rate of data transmission, but the sheer number of neurons and their richness of connectivity, compensates for the lack of speed. Indeed, it is the parallel behaviour of the neurons that makes the human brain a massively parallel system, and has become a focal point of AI research (McClelland 1987).

Attempts have been made to replace von Neumann's machine, with its passive memory, with a variety of architectures, which are perceived by their makers, to imitate the behaviour of the human brain. One development is the creation of associative memory, which is a combination of a large number of PEs that interact simultaneously. The

---

[1] It has been suggested that the amount of connectivity in brain, in comparison to the number of neurons, is very low (Fogelman-Soulie 1990).

contents of the memory are changed either by forming new hardware connections (eg, NETL : Fahlman 1979), or by changing the strength of existing connections (see section 3.41 for more information on neural nets).

Although the computational models vary from one machine to another, they all have to explore knowledge from a particular domain, perform certain manipulations and modifications, and finally reach certain conclusions. The structure of knowledge can be based on a particular mathematical formalism, e.g. Predicate calculus, or can use a more heuristic approach, such as semantic networks or frame-based networks (see chapter 2).

In the following sections a brief review of the classification of parallel architectures is given. After this, there is a discussion of the taxonomy of AI machines, which have certain characteristics relevant to the Sheffield Machine ("SM"). This is followed by a discussion of the architectural structure of the SM. The chapter ends with a review of some of the existing parallel architectures, setting out any comparisons to the SM.

3.2 TAXONOMY OF PARALLEL ARCHITECTURE MACHINES

Machines with parallel architectures can be classified according to the following criteria :

a) physical structures (Delgado-Frias 1987b).
b) Flynn's taxonomy (Flynn 1972).
c) Computational models (Hwang 1987).

In the first classification, there are different types of parallelism that may broadly be grouped into four categories; pipelined, bus oriented, switched network and array/tree based machines.

The pipelined system works on the basis of partitioning the tasks into several independent sections, which are executed one after another for a single task. A number of tasks may then be executed in parallel, each one requiring successively different sections of the same pipeline. There are, therefore, a number of PEs employed to provide such simultaneous execution of programs. A working example of the pipelined system is the Manchester Data Flow machine (Harrison 1986). Bus oriented and switched networks, often share the property of having a common memory, which may be accessed by each PE. Alice (Darlington 1983) at Imperial College is an example. The fourth type of parallel architectures is the array/tree based machine. The SM can be classified as a member of this family. In this type of architecture, the number of PEs and their physical interconnections vary from one machine to another. Thus, any

member of the family of array/tree based machines, can be associated with one of the following classes of architectures (these classifications are made with respect to the number of PEs 'n' employed in the system) :

a) fine-grained architectures; with a large number of simple PEs, eg NETL (Fahlman 1979), Connection Machine (Hillis 1985), Boltzmann Machine (Fahlman 1985) with the number of PEs > 10,000.
b) medium-grained architectures; average number of PEs are used, eg SM (described in detail in the following sub-section), Zmob (Bane 1981) with $100 < n < 10{,}000$.
c) coarse-grained architectures; small number of fairly powerful PEs are used ($n < 100$), eg Alice using five Transputers in each PE.

Flynn's taxonomy was based on machines using single or multiple streams of data and instructions. SISD (Single Instruction, Single Data stream) is for conventional von Neumann architecture. MISD (Multiple Instruction, Single Data stream) machines have not yet been built. In a SIMD machine (Single Instruction, Multiple Data stream), many processors simultaneously execute the same instructions but on different data, whereas, in a MIMD machine (Multiple Instruction, Multiple Data stream) there are several independent PEs, where each PE executes its own individual data.

In contrast to the above classifications, which were made in accordance with hardware specifications, the taxonomy of AI machines is made with respect to their computational models (Treleaven 1986, Hwang 1987). There are three major classes of AI machines :

1) language based machines; which are themselves subdivided into three groups :
   a) Lisp machines (or list processing machines),eg Symbolics 3600 series (Moon 1985).
   b) Prolog machines, eg PIE (Fuchi 1983).
   c) Functional programming machines, eg Alice (Darlington 1983).

2) knowledge based machines. This class of AI machines are subdivided into three groups as the following :
   a) Associative Networks/neural networks, eg NETL (Fahlman 1979), Connection Machine (Hillis 1985), SNAP (Moldovan 1985), Apriary (Hewitt 1980), Boltzmann Machines (Hinton 1984) and other neural nets (Aleksander 1990).
   b) Rule based, eg DADO (Stolfo 1987), NO-VAN (Boyle 1983).

c) Object based, eg SOAR (Hwang 1987), FAIM (Davis 1985).

3) Intelligent Interface Machines. In this class, machines are used in Speech Recognition, eg HEARSAY-II (Erman 1980), Pattern Recognition/ Image Processing, eg ZMOB (Minker 1983) and Computer Vision, eg Butterfly (Harrison 1986).

With this type of classification of AI machines, the SM, would fall into the second category; Knowledge based machines, and it would relate to the associative networks sub-division.In the following subsections, this category will be examined, with some examples.

3.3 KNOWLEDGE BASED MACHINES

The central objective of knowledge oriented machines is to efficiently explore and manipulate the powerful models employed for knowledge representation. Examples of these models are; semantic networks, frame-based networks, rule-based systems, neural networks and object-oriented systems (see chapter 2).

Object-oriented systems, such as SOAR (Hwang 1987) and FAME (Davis 1985), are characterised by the encapsulation of data together with all the procedures that manipulate it, into a uniform type; the object. Objects can only interact with one another by sending messages. Object-oriented systems have strong inner-relationships, as do frame-based systems. But in a pure object-oriented approach, data from a frame (eg, slot definition and values) must be kept within the object itself, and is not accessible from outside. Only the functions and procedures operating within the object have free access to the data. In practice however, the object-oriented approach utilises certain characteristics of both frame-based systems (i.e., representing objects by frames and performing inheritance operations), and of rule-based systems (i.e., using rules for inference).

In the remainder of this section, some of the associative network-based architectures; the SM, NETL system, Connection Machine ("CM") and Boltzmann machine together with a brief review of rule-based machines; DADO and NO-VAN systems are described.

3.4 ASSOCIATIVE NETWORKS MODELS

In a network of hardware units (Processing Elements, "PE"), there are several ways to implement associative networks (semantic networks, frame-based networks and neural networks are regarded as associative networks). These approaches are directly related to the granularity of objects and the PEs containing them, the objects being

linked to each other according to their relationships. Here, granularity is used to refer to the size of objects and PEs that are representing them.

One approach is to represent each node of a network by a simple PE, and use hardware links to represent the network's connections (eg, NETL). This approach is analogous to the addressing mechanism used in current digital computers. An address (a pointer) is used to link a data structure to another data structure. Fahlman (Fahlman 1979, 1981) and Minsky (Minsky 1980) suggested the replacement of addresses by real hardware connections. This removes the need for an addressing mechanism. Fahlman proposes a switching net which he hopes will overcome the connectivity problem (Fahlman 1987).

A second possibility is to employ large patterns of activities (a set of simple PEs with their physical links), to represent concepts, and data structures to be stored by modifying the interactions between these patterns (Feldman 1985). In the third approach, more complex PEs are employed to represent concepts and the relationships between them can be predefined or configured dynamically (eg, SM and SNAP).

These approaches treat objects in two different ways. In a frame-based system for example, objects or concepts are represented in symbolic form, whereas in neural networks a concept is represented as a pattern of activity (Feldman 1985). In symbol processing systems (eg production systems), the internal structure of the symbol is thought to be irrelevant to the way it interacts with other symbols, that is, the symbol has an identity, such as a unique character string. It is then compared to other symbols to determine whether it is the same or not, and the meaning of the symbol is determined by certain rules or programs that contain it, rather than by its own internal structure. In neural networks on the other hand, symbols have internal structures and interactions between symbols are determined by their internal structures. This property of neural representation provides distributed control and a better environment for parallelism which avoids the overheads of an external controller and internal communications. This concept is highlighted in chapter 6 where the test results show the effect of overheads caused by communications.

3.41 A BRIEF DISCUSSION ON NEURAL NETS

Neural networks refer to a certain class of massively parallel fine-grained architectures (Feldman 1985, Rumelhart 1987). They use a large number of PEs, each connected to a number of other PEs. The PEs are very simple and have little information stored internally, and are regarded as a kind of short-term working memory. The long-term storage is accomplished by altering the pattern of

interconnections among the PEs, or by modifying a quantity called weight, associated with each connection. This method of connection, as the principal means of storing the information, has led to it being labelled, by some researchers, as connectionism (Feldman 85).

In neural computing, the approach is to implement those principles that are perceived to be the same as in our brains. There is no doubt that the human brain and its complicated structure is poorly understood. Nevertheless, there is a basic understanding, at a low level, of how a brain works. There are approximately, as mentioned above, 10^{12} neurons in human brains where each neuron has connections, between 10^3 to 10^4, with other neurons (Beale 1990). The neuron is the basic PE in the brain and form two main types; local processing "interneuron" cells and "output cells". The interneuron cells have their input and output connections over 100 microns, whilst the output cells connect different regions of the brain to each other, connect the brain to muscle, or connect from sensory organs into the brain (Feldman 85). Although the basic operation of neurons is clear, on a microscopic level its operation is complicated and not fully understood (McClelland 87). In figure 3.1, a representation of the basic features of a neuron is shown.

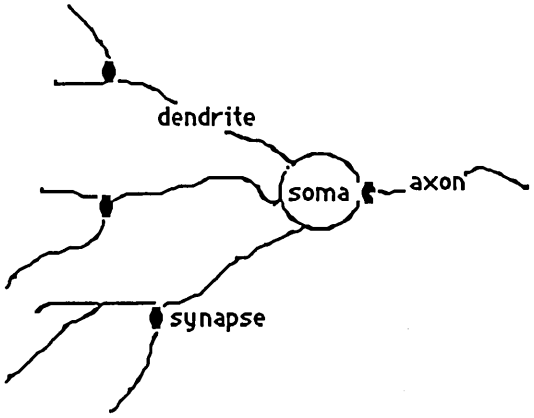


Figure 3.1, basic features of a biological neuron.

Note that the diagram shown in figure 3.1, is basic and simple; for a more detailed definition refer to the following : Beale 1990, Aleksander 1990, Rumelhart 1987, McClellend 1987 and Feldman 1985. The overall definition of neuron's operation is given below, which has been adopted from the references given above.

The neuron receives many inputs which are added up in certain ways. The neuron will fire if the sum of inputs has met the required condition, otherwise the neuron will stay inactive. Dendrites act as the links through which all the inputs arrive and, are attached to the soma. The soma is the cell body. In addition to dendrites, there is another type of link (or nerve process) attached to the soma, called an axon, and this serves as the output link of the neuron. The axon is electrically active and is a non-

linear threshold device that produces a pulse (1 Milisec duration). The axon is attached to a dendrite of another neuron by a synapse. The junction between an axon and a dendrite is not directly linked. At this junction, a synapse releases a certain chemical when its potential is raised sufficiently by the pulse it has received from its connected axon. The chemical released provides a temporary link with the dendrite, and from this link electrically charged ions flow towards the next cell through its dendrite. On each dendrite there are many synapses, some of which may be active at a particular time, and, If the amount of electrical pulses have reached a certain threshold, the neuron will fire. Note that axons are often absent from interneurons, which have both inputs and outputs on dendrites.

The most important aspect of artificial neural nets is their learning capability, which is based on the behaviour of biological neurons. Learning is thought to occur when changes are made to the connections between cells at synaptic junctions. These modifications will either reinforce the connection or reduce the coupling effect. One method of providing this kind of adjustment in artificial neural nets, is the introduction of weighting, which varies within a certain boundary. The weight at the higher level of the range represents a stronger connection, in contrast to its lower level values.

The basic operation of an artificial neuron is to add up its inputs and to produce an output, if this sum is greater than some value called the threshold value. The neuron receives inputs via dendrites, which are connected to the outputs from other neurons by synaptic junctions. At these junctions, the effectiveness of the signals are adjusted according to the strength of the coupling. The cell body receives all these inputs and fires if their total is greater than the threshold value.

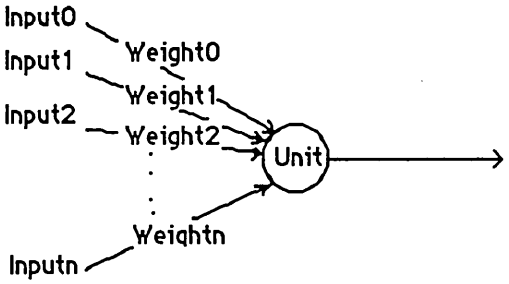


Figure 3.2, McCulloch-Pitts neuron.

The operation in a basic model of an artificial neuron involves adding up the sum of inputs and comparing it with the internal threshold level. If this level is exceeded, the neuron will turn on, otherwise it will stay off. This is roughly the definition of the model neuron that was proposed by McCulloch and Pitts in 1943. Rosenblatt, in 1962, called these neurons "perceptrons" and pioneered an appropriate method of

simulating them on digital computers (Rumelhart 1987). Figure 3.2 shows a model neuron proposed by McCulloch-Pitts.

A learning approach adopted for the type of neuron shown in figure 3.2, is to set up the neuron with random weights on its input lines. This can be regarded as the starting state, where the neuron knows "nothing". The object can then be presented to the neuron (eg; letter A). After adding the inputs and comparing it to the threshold value, if the result exceeds the threshold, the output will be 1, otherwise 0. It is only in the case of output being 0 that certain adjustment should be done. The adjustment here, would be to increase the weights so that when shown a letter A, the sum of all the weights will exceed the threshold and the neuron will output 1.

The perceptrons were introduced as a single layer neural nets. That is, it has only one node between any overall input and overall output. There are many different topologies for the shape of neural nets; feed-forward (associative) nets that includes both single-layer and multi-layer nets; and feedback nets (Alekssander 1990).

3.42 THE NETL SYSTEM

In 1979, Fahlman developed the NETL knowledge based system, with an architecture that is classified as a massively parallel fine-grained architecture (Fahlman 1979). In the NETL system, the semantic network is implemented in hardware and there is no clear division between the knowledge representation model and its architecture. However, in this subsection, an attempt has been made to examine the whole system with the emphasis on the architecture of the NETL system. The architecture is basically an active semantic network memory; nodes represent concepts, and links represent relationships between these concepts.

Each element (PE) contains only a few bits of memory. Fahlman suggests that, by propagating markers through the system, a variety of simple searches, set intersections, and the inheritance of properties throughout the hierarchy, may be performed. The NETL system is controlled by an external serial machine of the conventional type. The knowledge in NETL system is stored not in passive memory cells, but in the pattern of active interconnections among the PEs.

Fahlman has suggested the development of a one million elements system, where there are six million link-wires to a possible one million destinations (Fahlman 1985b).

3.421 THE PARALLEL NETWORK SYSTEM

Fahlman based his knowledge representation scheme on type hierarchies and inheritance of properties. There are two types of nodes to represent concepts; individual nodes (INDV-Node) for individual entities, and Type-Nodes for class descriptions, from which any number of individual copies can be made. Each type node is associated with a particular set, which itself is represented by an individual node, known as a Set-Node.

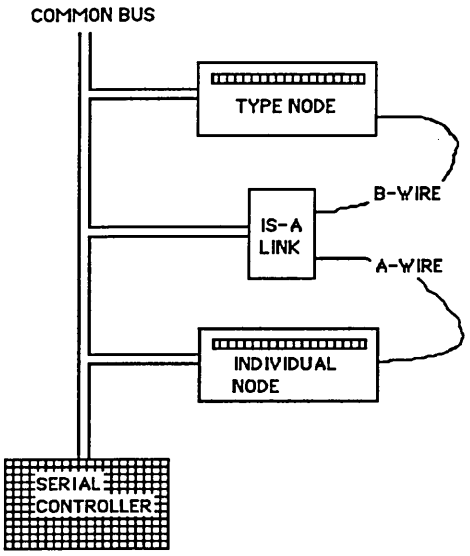


Figure 3.3, the basic hardware components in NETL.

The basic components of the NETL system are shown in figure 3.3. Concepts are represented by simple PEs called nodes, and relations among these concepts are represented by additional PEs called links. The nodes and links are all attached via a common bus to an external serial computer that plays the role of controller to the whole system. Serial numbers or unique names are used to represent nodes and links in the system. The controller can broadcast simple commands to all the nodes and links or to an individual node or a link by issuing their relevant serial numbers. In return each individual node or link can communicate with the controller by sending its serial number. There is a queuing mechanism used which can be employed when more than one PE is trying to contact the controller, at any time (Fahlman 1985).

Each node contains its serial number, an optional name, information about the type of concept that it is representing, and a small number of flip-flops which can be used to mark the node. The number of marker bits was suggested to be 16 (Fahlman 1979), but elsewhere Fahlman has increased this number to 18 (Fahlman 1985b).

The element representing each link is a simple PE as well. It contains a few bits of type information that specify the link type, and a variable number of wires that can be

connected to various node terminals (currently there are 6 wires). In figure 3.3 an IS-A link is shown; it has type bits indicating that it is of an IS-A type, and two wires that are connected to nodes in the network; wires A and B are connected to an individual node and a type node respectively, which can then be read as "individual node" IS-A of the type "type node."

3.422 CREATING DESCRIPTIONS

A knowledge pattern may be regarded as a description which defines a concept and its associated properties. This consists of a base-node (an individual node or a type node) representing the concept, and a set of role-nodes, connected to the base-node by various links.

To create a simple description, four types of nodes may be employed; a type-node to define the class, an individual node or role node, and a map node which can be used to transfer all the definitions of the class to a lower level individual node. An individual node may be created by finding a physical-individual-node to represent the concept of that particular Ind-Node, and another node representing the context. Obviously, all the nodes that define an individual concept are connected to the higher levels of the hierarchy.

To create a type description, a Type-Node in conjunction with a Set-Node are created. The Set-Node is created first. This is done by the controller, finding an unused Indv-Node and then connecting it to the Set-Type-Node. An unused element (a Type-Node) should be found to create a Type-Node, then connected to its parent node and the Set-Node. Fahlman uses statements (for links) to define relationships between concepts and the information related to them. Some statements are represented as link statements, others are represented as individual-statement-descriptions which describe an IST-node (Individual-STatement-node). The statement itself is represented by the handle-node of the link element or IST-node. As with any other elements used in the system, a statement can be treated as an object with properties and class-membership of its own. A statement can have a default value which can be overridden, or have a specific value which cannot be overridden. All these constraints are attached to the handle node or IST-node. The functions of a handle node is similar to that of facets in frames (see chapter 2).

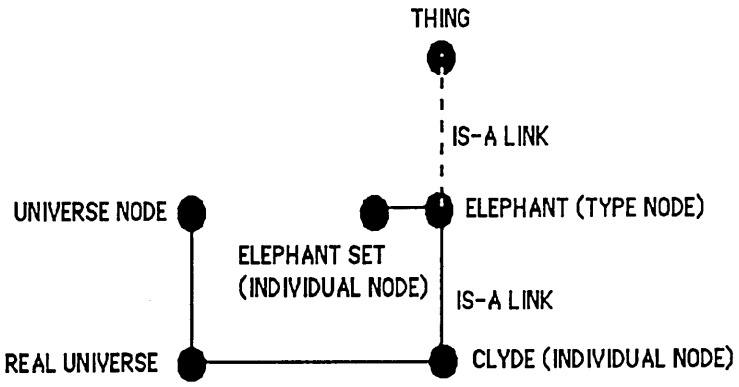


Figure 3.4, a description for an individual concept; Clyde.

Figure 3.4 shows a description for the famous American elephant Clyde. To create a description for Clyde, an unused individual node is selected to represent the concept Clyde. A link element is chosen to represent the link between Clyde and its type-node; Elephant. Fahlman uses a VC link (Virtual Copy link) to connect an individual node to a type node which can later be used to inherit the description from the type node (Fahlman 79). By assuming the concept Clyde exists in the real universe (real-u in NETL), an existence wire (another link element) is used to connect Clyde to the Real-universe node.

3.423 INHERITANCE AND MARKER BIT PROPAGATION

Each element has about 15 marker-bits, which are generally used for inference. The virtual copy (VC) link is one of the link wires which Fahlman uses for connecting two concepts and the consequent inheritance of properties, from one concept to the other. In general, VC links may perform two tasks. A VC link copies all the description from the Type-Node to the lower Type-Node or an Indv-Node. It also connects the indv-node/type-node to the set associated to the original concept. In other words, the Set-Node connected to the original type-node would be inherited to the lower type-node or Indv-Node. Further, properties inherited through the VC link are the role-nodes connected to the original Type-Node. When any particular information, that is specific only to the newly created Indv-Node, is added to the system, a map-node is employed, which represents the new information and the inherited properties (usually map-nodes are used to transfer the general attributes from higher levels to the individual node).

Fahlman introduced VC links to connect a child-node to its parent-node and also to provide a path for the child to inherit description from its parent(Fahlman 79). In a recent publication he calls such links "IS-A" links (Fahlman 1985b) and defines an IS-A hierarchy.

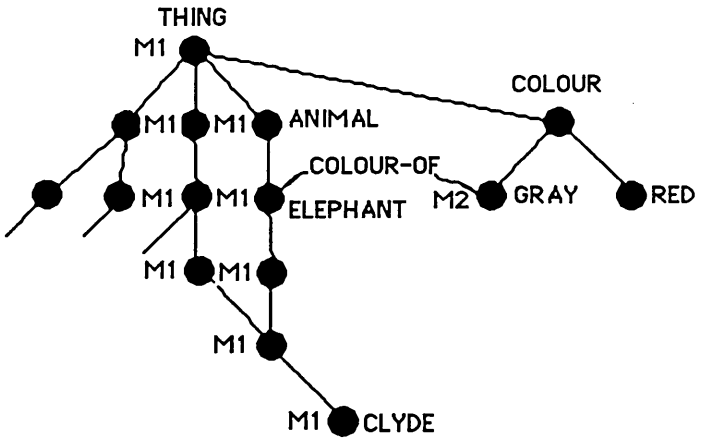


Figure 3.5, an example of marker propagation.

These complex structuring rules allow specific queries to be easily implemented by marker propagation. For example, when a query "what colour is Clyde " is made, the controller tells Clyde node to set marker bit M1. In the next broadcast, all the IS-A links that contain marker M1 on their A wire nodes are instructed to place marker M1 on their B wire nodes (see figure 3.5). In this cycle (or operation), all the nodes that are one level above Clyde, are now marked. This process is repeated until all the relevant nodes to Clyde, in the IS-A hierarchy, are marked. The controller then broadcasts to all the colour-of links with marker M1 on their A wire, to set marker M2 in their B wire nodes. Finally the controller instructs all the nodes with marker M2, to report to it via the common bus. If the knowledge has been entered properly and Clyde has just one colour, then this will apply to just one node, which will report in.

Set intersection is another method of inference employed in Fahlman's NETL system, which is similar to Quillian's marker propagation (Quillian 1968). Two markers are sent down from two different hierarchies to mark the relationship specified in the query. The controller will then issue a command which asks any relationship statement, that has marker M1 on one side and marker M2 on the other side, to report in.

Fahlman uses a very complex knowledge representation formalism, which has a small granularity in comparison with that of Frames. That is, a single processor represents a much smaller piece of the overall knowledge. He claims that his formalism may be used in serial machines, but in a knowledge base of any significant size, propagation time with such a small granularity might be expected to make access time very low.

3.43 THE CONNECTION MACHINE

Although Hillis's Connection Machine (CM) was initially designed for AI applications, its latest version; CM-2, is now available for large scientific and commercial applications. CM-2 comprises 4K-64K PEs, where the size of memory of each PE 8K-128K bytes (see figure 3.6).

There are 16 PEs mounted on a single chip, together with a router unit. The router unit is the basic component in the packet unit communication network, providing communication between the chips themselves, between the chips and the host (main controller) via the bus, and between the PEs and the control unit mounted on a single chip.

There are three levels of control mechanism in CM :

a) the host (a serial machine).
b) microcontrollers, one per chip of 16 PEs.
c) the PEs.

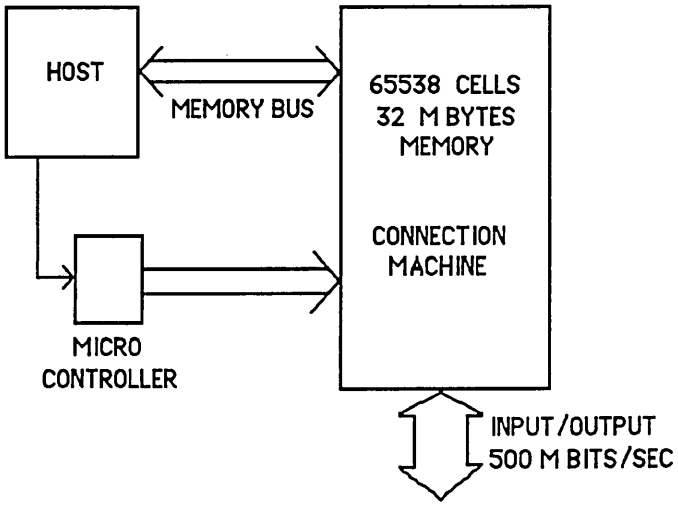


Figure 3.6, the architecture of the Connection Machine.

All PEs execute instructions from a single stream in the SIMD manner generated by the micro controller of every chip under the direction of the host. Further, the PEs can retrieve data from the external memory to perform manipulations, and transmit the results back to the main memory.

The communications among PEs on a chip are provided by routers (which can also contact the other routers on other chips). There are three parts to each router :

1) The injection, which transmits new messages into the network.
2) The heart, which propagates messages among chips.

3) The ejector, which receives and transmits messages to the appropriate PE.
There are also local communications within each chip without involving routers, by which each PE is linked to its, north, south, east, and west neighbours.

Within the three layers of the CM, there are various types of instructions. Host instructions, executed by the host, in turn produce macroinstructions, that are interpreted by the microcontroller to produce microinstructions. Subsequently, microinstructions are executed by the microcontroller to produce nanoinstructions, to be executed by the individual PEs.

The initial design of the CM was aimed at a hardware implementation of the semantic network (Delgado-Frias 1987a). It has turned out now that the resulting architecture is more general purpose than it was designed for. The CM is mostly used for commercial and scientific applications such as high speed document retrieval from a large data base and it has been used for memory based reasoning and natural language processing (Stanfill 1986, Durham 1987).

With reference to the above executional hierarchies, it seems that the layers are introduced as a trade-off between the simplicity of each PE's structure on the one hand, and the communication overheads that have resulted from having a number of PEs mounted on a single chip on the other hand. Hillis emphasises the high granularity of the CM. The CM has 4K-64K PEs, with each PE being so simple that it cannot on its own perform any meaningful computations. Consequently, there are in place a large number of physical connections, and layers of control mechanisms, which can be regarded as an extra burden on the overall, and local, control systems.

3.44 THE BOLTZMANN MACHINES

This type of machine consists of massively parallel networks of simple PEs, called "units" (Fahlman 1985b, Trleaven 1986). These units are logically simple computing elements, connected to each other with bidirectional links (see figure 3.7).

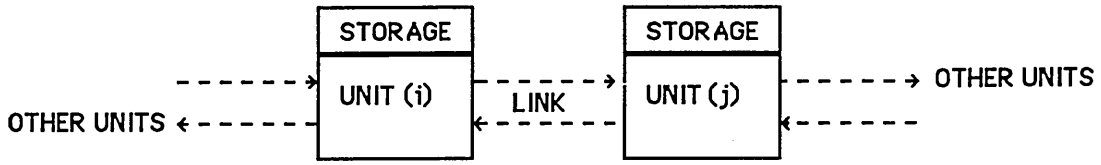


Figure 3.7, the basic physical units in Boltzmann machine.

Each link has a weight associated with it, emphasising the strength of the interrelation that it represents. The links themselves represent a so called "weak" pairwise constraint between the hypotheses. A positive value weight for a link indicates that the

two hypotheses support each other, whereas a negative weight suggests that both hypotheses should not be accepted.

In a Boltzmann machine each unit is either on or off. The resulting structure of all the units with their on/off states is a machine in a certain state at any time. This state can be given a value, which is the sum of the weights of all the "on" units in the system, plus a threshold. This value is termed the "energy" of the system at that particular state. Thus, by giving the system a set of hypotheses, some units will form particular states representing these hypotheses. In other words, the system is given a certain energy. Once the system has gained energy, after certain manipulations and interpretation of the hypotheses, these hypotheses can become knowledge themselves.

At Carnegie-Mellon University, the initial version of the Boltzmann Machine was known as Thistile (Fahlman 1985b). In the Thistile system, value-passing and marker-passing are combined as a single method of communication. Fahlman has suggested a method of classifying massively parallel machines, by distinguishing the type of signal that is passed among PEs, as follows :

a) Message passing systems, eg SM.
b) Marker passing systems, eg NETL .
c) Value passing systems, eg traditional analog computers, passing continuous quantities of numbers.

However, the Boltzmann machine is a special type of neural net and it has some similarities to the McCulloch and Pits model discussed above. In the network, each neuron has two states : the output is '0' and the neuron will not fire, or the output is '1' and the neuron will fire, where the inputs to each node (neuron) are from other neurons in the net.

3.45 SNAP

At the University of Southern California, SNAP (Semantic Network Array Processor) was developed to map and manipulate semantic networks (Dixit 1984, Moldovan 1985). The SNAP architecture consists of a 2-D array of homogeneous PEs connected to each other locally and globally (see figure 3.8). The local communication is based on nearest neighbour connection and global communication is provided by columns of buses. Message passing is used for communication, where messages are propagated through the array using both local and global communication paths (buses). A semantic network is mapped onto the system with nodes in PEs, and links between nodes stored in the pointer memory of each PE. SNAP is operated by a front-end controller, which

also interfaces the system to a host computer. Each PE has a unique address comprising a pair (row and column), and contains three main parts :

1) Content Addressable Memory (CAM).
2) Processing Unit (PU).
3) Communication Unit (CU).

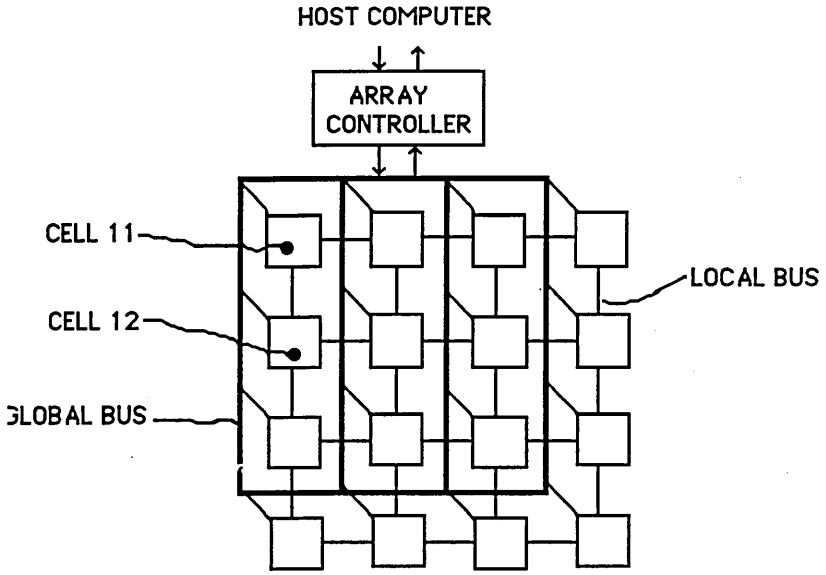


Figure 3.8, the architecture of SNAP.

The CAM consists of two parts: cell memory which is the local memory; and pointer memory, that contains relation names and their addresses. The PU, controls both CAM and CU. The PU has a set of reduced set of instructions namely AND, OR, SET, etc. The function of the CU is to perform data transfers between PEs. This is done in two phases:

1) Process-phase, pops the top packet of its FIFO (First In First Out) queue and places it in the packet register.
2) Send-phase, compares the address to the PE's name and sends the packet to the destination node.

3.5 RULE-BASED MODELS

In this section, rule-based oriented machines such as the DADO and NO-VAN are briefly examined. The algorithm for rule-based production systems has been reviewed in detail in chapter 2. The basic algorithm for the execution of production systems goes through a three-phase cycle.

First, the condition clauses of all rules are "matched" against the working memory, which represents the state of the world. One of the successfully matched rules is then "selected" according to some control strategy. Finally, in the "act" phase, the selected

rule is fired or executed, and the working memory is updated. Generally, this is done in serial fashion and is inherently very slow (Gupta 1985). DADO and NO-VAN are special architectures designed to speed up the execution of rule-based systems.

3.51 DADO Machine

DADO is a parallel, tree structured machine designed at Colombia University (Stolfo 1987).

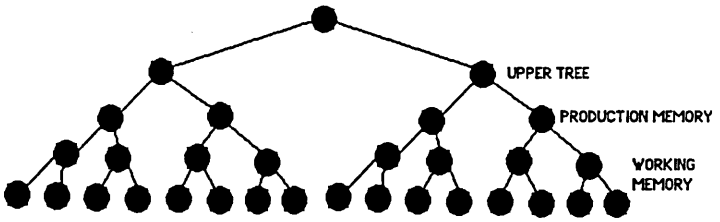


Figure 3.9, the architecture of DADO Machine.

The configuration of the DADO machine is based on a binary tree (see figure 3.9), and each node is a primitive PE comprising a processor, small local memory (20k bytes in DADO2 prototype), and an I/O (Input/output) switch. The DADO2 system consists of 1023 PEs, and it is hoped that a full-scale version would comprise a large number (in the order of thousands) of PEs. The PEs are interconnected in a complete binary tree.

Under the control of run-time software, each PE is capable of operating in two distinct modes. In the first mode, called SIMD (single instruction stream, multiple data stream), the PE executes instructions broadcast by some ancestor PE within the tree. In the second mode, MIMD (multiple instruction stream, multiple data stream), each PE executes instructions stored in its own RAM, independently of the other PEs. A single conventional host processor adjacent to the root of the DADO tree, controls the general operations of all the PEs. The study of DADO2 system performance, and its speed-up over VAX 11/750 (Stolfo 1987), showed that in the cases of executing certain production systems with a smaller number of rules, the speed-up was as much as 31 fold. Further, this study showed that, the more the number of rules are, the more execution time would be needed.

3.52 NO-VAN MACHINE

NOVAN (or NO-VAN) machine (Boyle 1985), is a large parallel active memory that consists of PEs, typically between 4K to 32K (with a projected increase to 1 million).There are three types of PEs :

1) Small PEs (SPEs),
2) Large PEs with disc units,
3) Large PE Network.

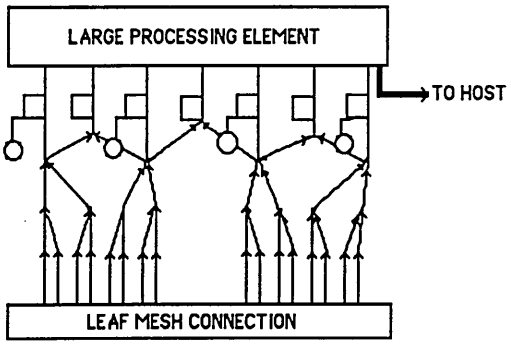


Figure 3.10, the hardware organisation of NO-VAN machine.

The small PEs are configured in the form of a binary tree, whose leaves are also interconnected, to form a 2-dimensional orthogonal mesh (see figure 3.10). Each Large PE in the network broadcasts instructions to its SPEs for simultaneous execution.

The architecture of a SPE consists of 256 Bytes of RAM, a Processor and an I/O (Input/Output) switch. A large PE comprises a 32-bit microprocessor (Motorola 68020) and a sufficient amount of RAM to hold the program (in NOVAN 1, VAX 11/750 is employed as a Large PE). Large PEs are connected by a Large PE network, with a high bandwidth, low latency two-stage interconnection scheme. The central point in the design of the NOVAN machine is to provide a single, flexible, general purpose AI machine.

3.6 THE SHEFFIELD MACHINE

The Sheffield Machine ("SM") was originally designed as a numerical data-flow architecture (for the modified version see Loh 1982a), consisting of a rectangular array of PEs with a supervisory system, or controller, being positioned at the centre of the array and being represented by nine PEs. The whole array is divided into a number of physical grids, each of which comprises nine PEs, with the top left hand side PE known as a recipient node. Each PE has four connection wires which are connected to its four neighbours. There is an extra connecting wire added to each recipient node, used to connect it to a switching network, which is then connected to the disk unit. There are two processors in each PE; one is to provide communications with other PEs and the controller, and the other is used for data manipulations. Another type of communication is performed between the switching network and the recipient nodes

to transfer data from the disk to each of the recipient nodes of the grids. The internal communication between PEs involves message passing using data packets.

Later developments of the simulated SM involved the simulation of the data-flow architecture suggested by Loh and Brown (Loh 1982b), with modifications to the system, eg. an actor of a data-flow network was represented by a single PE (Walker 1983), the array size was increased to (99 x 99), and the central grid was permanently allocated to the controller. The size of the data items were increased from a single variable to a variable length data structure (Green 1984).

In the further development of the SM, comparison was made with the NETL system (Fahlman 1979) to see if it was feasible to modify it so that it could be employed in AI applications. As a result of this modification a working simulation was produced and the notion of data-flow architecture, using Petri-Nets to represent programs and data, was replaced by a parallel architecture employing a limited knowledge representation scheme (Hird 1985). In recent developments on the SM simulation, an attempt was made to build a distributed supervisory system over the central grid (Hollinghum 1985), and a graphical display of the movements of data packets between the controller and PEs (White 1986).

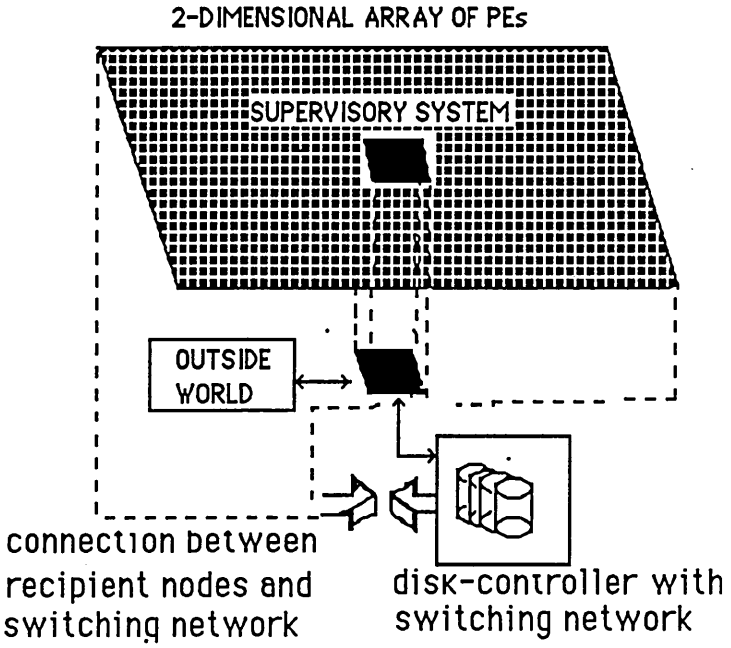


Figure 3.11 the SM architecture at the start of the project.

The version of the simulated SM architecture, at the start of this project (see figure 3.11), can be described as follows :

a) A rectangular array of a large number of PEs (100 x 100), each PE consists of one communication processor, one arithmetic processor (utilised for

knowledge manipulations), and two distinct memories for each processor, plus a shared memory (see figure 3.12).

b) Centralised supervisory system, comprises nine PEs (a logical grid).

c) Bidirectional links, connecting each PE to its four immediate neighbours .

d) Switching network, connecting all the recipient nodes to the secondary storage.

e) Approximately 1100 recipient nodes, each positioned at the top left hand side corner of each logical grid.

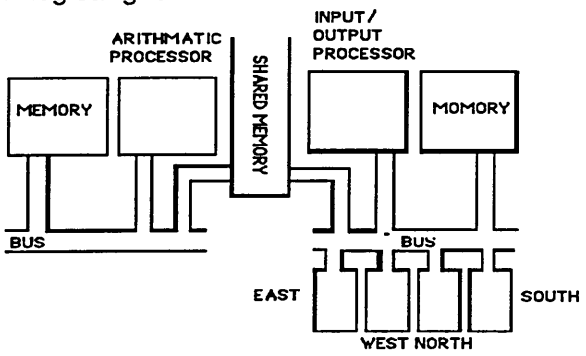


Figure 3.12, a Processing Element in the SM.

The centralised supervisory system is responsible for communication with PEs, issuing commands to PEs for knowledge manipulation, data packet creation and the control of their propagation, and the management of external communications with the outside world. All the recipient nodes are positioned at the top left hand side corner of each grid and connected to the switching network. Thus, relevant information can be transferred in parallel from disks to a subset of recipient nodes.

A frame-based knowledge representation language has, since the start of this project, been developed that is suitable for the SM architecture. In turn, the SM architecture has been modified and extended to reflect the computational requirements of this frame-based language.

Figure 3.13 shows the modified architecture with a lattice of buses. In this configuration, each PE is connected to the four buses that surround it (bus grid).

In each bus grid there are several switches that are used to divert the direction of communication path. The type of buses used are as follows :

a) Control buses: these are used by the controller to broadcast signals to all the relevant switches and PEs (the controller uses an address bus, control bus and an interrupt acknowledge line).

b) Common buses : these are used to provide one-to-many communication between PEs themselves, and between PEs and the controller.

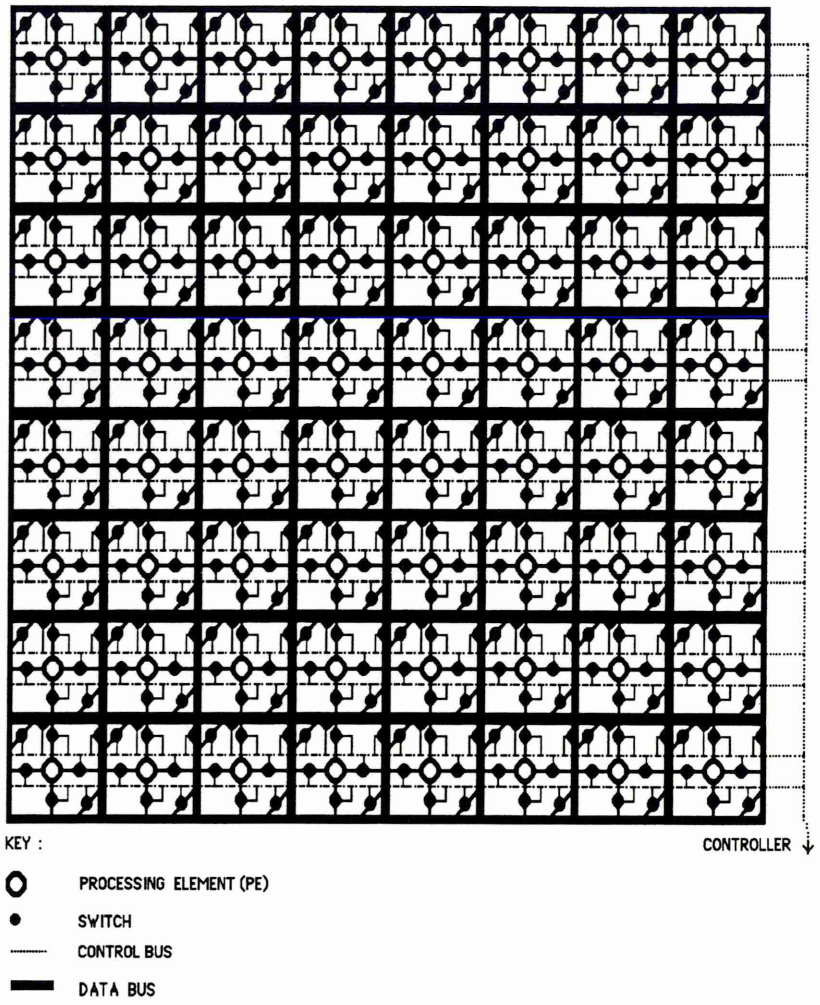


Figure 3.13, the new architecture of the SM.

There is another type of link; diagonal "row-to-column" link. It is used to join a horizontal bus to a vertical bus. In each bus grid, there is one diagonal "row-to-column" link at the top left corner, and another one at the bottom right corner.

Switches are positioned as follows :

1- There are four switches that are positioned at (NSEW) points of each PE. They are used to link the PE to appropriate buses.

2- There is a switch positioned on the horizontal bus above each PE. This switch is used to stop signal being transmitted any further.

3- There is a switch positioned on the vertical bus on the right hand side of each PE. This switch is used in the same manner as in number 2.

4- There is a switch on each diagonal "row-to-column" link. When this is closed, an L shape bus will be configured.

By opening or closing appropriate switches, the operation of a parallel dynamic hardware configuration suitable to the logical configuration of a relevant hierarchy is

performed. This is a critical feature, where the time spent on configuration will not exceed the disk access/transfer time.

3.61 FINAL REVIEW OF THE SM'S PARALLEL ARCHITECTURE

The SM is of type medium grained MIMD parallel machine (Flynn 1972) and may be classified as a knowledge based machine of type associative model (Hwang 1987).

In the SM, in order to estimate the aggregate memory, the aggregate communication bandwidth, and the overall performance with respect to the the execution of instructions per second, the following assumptions have been made :

-The maximum number for the PEs in the array is assumed to be 33 x 33. In the rectangular array, PEs are connected via tri-state drivers to a lattice of horizontal and vertical buses. Each PE is assumed to be a 32-bit microprocessor, with 33 MHz clock speed (1 cycle takes approx. 30 nsecs), and of 16 Kbyte of memory. The aggregate memory of the SM (distributed) will then be 17 Mbytes (approx). Note that the memory in each PE can be expanded substantially, eg; 8-16 Mbytes.

-There are approximately 33 vertical buses and 33 horizontal buses, with 33 PEs connections per bus. The inter-communication is provided by a lattice of 32 bit buses (with tri-state switches). The communication bandwidth for each PE in the SM with a 33 MHz clock speed, is 2.5 Mbyte/s and the aggregate bandwidth for the whole machine is 2.7 Gbyte/s.

-The performance of each PE in the SM, with 33 MHz clock speed, is assumed to be 4 Mips. Note that this estimation is for CISC (Complex Instruction set Computers) chips, where in a RISC (Reduced Instruction set Computers) chip, the time taken for each instruction may take fewer cycles and thus increasing the performance. The aggregate performance of the SM is 4.4 Gips

-As mentioned before, the knowledge base of the SM is too large and requires a disk unit with a large capacity and a very high transfer rate. These requirements can be achieved by commercially available disk subsystems that operate in parallel. In DIA (Disk In Array) for example, a number of magnetic disk drives, configured into an array and run in parallel, achieve a fast access time of 10 mseconds and a high transfer rate of 36 Mbyte/s (Oyama 1991). This subsystem has the storage capacity of 15 Gbytes, which can be increased to 120 Gbytes by adding extra disk drives to it. In chapter 5 disk arrays and their benefits for the SM will be discussed in more detail.

A switching network may be employed in the SM, which can then be set by the controller to assist the mapping operation from the disk unit to the PEs in the array. A 128 way dynamic crossbar switch can provide a bandwidth of 100 Mbyte/s (Hayes 1988).

To conclude, looking at the spectrum of granularity, it seems clear that there is a trade-off between the high granularity (number of PEs) and the complexity of each individual PE. An example of this is the comparison between the complexity of PEs in NETL, and that of PEs in NOVAN or Alice machine (Darlington 1983). This may be reflected in the communication constraints as well, communication being quite intensive and lengthy between two complex PEs in the SM, and light and infrequent on the other machines. This notion can be exemplified by message passing in the SM or SNAP, and bit propagation in NETL. In figure 3.14, a resume of comparison between the SM and some of machines belonging to the knowledge based stream is given.

Machine	Knowledge Rep. Formalism	Communication	Structure of each PE	Connections amongst PEs
SM	Frame-based net	Message passing	Complex	1 PE to 4 PEs
NETL	Semantic net	Marker Passing	Very Simple	1 PE to Many PEs
Bolts. M	Associative net	Marker Passing	Very Simple	1 PE to Many PEs
C M	Semantic net	Bus-oriented	Simple	1 PE to 9 PEs
SNAP	Semantic Net	Message Passing	Simple	1 PE to 4 PEs
DADO	Rule-based	Bus-oriented	Simple	Binary Tree Form
NOVAN	Rule-based	Bus-oriented	Simple/Complex	Tree Form

Figure 3.14, characteristics of some of the knowledge based machines.

3.62 COMPARISON WITH THE EXISTING PARALLEL MACHINES

In this section, a brief review of some of the parallel machines that have already been developed and are used in industry is given. The purpose of the information[2] given here is mainly to highlight the main characteristics of these machines that can be compared with that of the SM.

---

2 This information has been collected by contacting the manufacturers and their sale-publication.

Machine : AMT (DAP) (Active Memory Technology Ltd)

Type : Fine grained SIMD array

PEs, Number, Memory : single bit processor, 1024-4096, 32-1024 kbit memory

Memory Shared : none

Inter-Comms : Nearest Neighbour and Orthogonal buses

Input/Output : SCSI or DEC link at app. 2 Mbyte/s

Disk connection : direct at 16 Mbyte/s

Performance : 1130 MIPS (integer), 3.1 GIPS (logical AND), 28-108 MFLPS (real)

Operating system and langs : provided by the host machine (VMS or UNIX), Fortran and C

Host : Vax or Sun workstation

Machine : MasPar MP-1 (MasPar Computer Corporation)

Type : Fine grained SIMD array

PEs, Number, Memory : Custom designed chips with 32 PEs/chip, 1024-16384, 16Kbyte

Memory Shared : none

Inter-Comms : Nearest neighbour and Global router

Performance : 26 GIPS (integer), 1.3 GFLOPS (real)

Operating system and langs : ULTREX, extended version of C

Host : Vaxstation 3520

Machine : TMC CM-2 (Thinking Machine Corporation)

Type : Fine grained SIMD array

PEs, Number, Memory : Single bit, 4096-64536, 64-1024 Kbit

Memory shared : none

Inter-Comms : nearest neighbour, router communication based on hypercube communication network

Input/Output : to disk farm (max 50 Mbytes/s

Performance : peak performance on the largest CM2 is 28 GFLOPS (the highest recorded : 5.6 GFLOPS)

Operating system and langs : from the host system; UNIX or Lisp, Fortran 90, C and Lisp

Host : Sun 4 workstation, Vax or Symbolics 3600-series

SHARED MEMORY MULTIPROCESSORS

Machine : Alliant FX/2800 (Alliant Computer Systems)

Type : Symmetric vector multiprocessor with shared memory

PEs, Number, Memory : i860, 8-28, 1 Gbyte

Inter-Comms : through the shared memory

Input/Output : through LAN

Performance : peak; 1.12 GFLOPS

Operating system and langs : extended versions of UNIX, different version of Fortran, Ada, C

Machine : BBN ACI TC2000 (BBN Advanced Computers Inc)

Type : Coarse grained, shared memory MIMD

PEs, Number, Memory : Motorola 88100 RISC,16-32, each node has 48 Kbyte of cache and 4-16 Mbyte DRAM

Inter-Comms : Butterfly switch provides point to point inter-processor communication

Input/Output : 16-32 serial ports and Ethernet

Performance : peak; 960 Whetstone MIPS

Operating system and langs : extended versions of UNIX, Fortran, Ada, C

Machine : Convex C2 (Convex Computer Corporation)

Type : Multiprocessor, tightly coupled shared memory

PEs, Number, Memory : 64 bit, 1-4, 2 Gbyte 64-way max interleaved

Inter-Comms : 64 bus with max 200 Mbyte/s

Input/Output : Ethernet

Performance : peak; 200 MFLOPS on a 32 bit Whetstone benchmark

Operating system and langs : Convex UNIX, CCC, Fortran 77, Ada

HYPERCUBES : DISTRIBUTED MEMORY, MIMD PARALLEL COMPUTERS

Machine : iPSC/860 (Intel Scientific Computers)

Type : Medium grained MIMD hypercube system

PEs, Number, Memory : 80680/80386, 8-128, 8-16 Mbyte/node

Inter-Comms : each processing node has a direct-connect message routing chip

Input/Output : SCSI supports seven disk drives with 4 Mbytes/s data transfer rate

Performance : 60 MFLOPS/node, total of 7.6 GFOLPS (peak)

Operating system and langs : host; UNIX, nodes; NX/2, Fortran 77, C and Ada

host : Intel 80386 based PC-type

Machine : NCUBE-2 (NCUBE Corporation)

Type : Medium grained MIMD hypercube system

PEs, Number, Memory : 64-8192, Custom chip, 1-64 Mbyte/node

Inter-Comms : each processing node has 14 serial links out of which 13 are used for inter-comm

Input/Output : max 1024 I/O nodes

Performance : 7.5 MFLOPS/node, total of 27 GFOLPS (peak)

Operating system and langs : Vertex, UNIX, EXPRESS, Fortran 77, C

host : Sun workstation, DEC Vaxstation

TRANSPUTER-BASED PARALLEL MACHINES

Machine : Caplin HEX (Caplin Cybernetics)

Type : Transputer based Micro Vax add-on boards

PEs, Number, Memory : T800s, 8 upward, 64Mbyte/board

Inter-Comms : inter-board communication, peak 3 Mbit/s

Input/Output : Via Q-bus (on board) to Micro Vax

Performance : max 200 MIPS (16 node system)

Operating system and langs : Vax/VMS or ULTRIX, Fortran, C, Pascal, Ada and Occam

host : Vax/Micro Vax

Machine : Meiko Computing Surface (Meiko Scientific Ltd)

Type : Distributed Memory Reconfigurable Multiprocessor

PEs, Number, Memory : T800s/i860, 1-100s, 0.5-32 Mbyte/T800

Inter-Comms : inter-board communication with 20 MHz bandwidth

Input/Output : SCSI interfaces

Performance : 1 MFLOPS/T800

Operating system and langs : SunOS derivative, Fortran, C, Ada, Lisp and Occam

host : Self-hosting or hasted by Vax or Sun

Machine : SuperNode 1000 (Parsys Ltd)

Type : Reconfigurable Transputer network

PEs, Number, Memory : 16 T800s in a SuperNode (min), 16-64, 16 Mbyte DRAM or 256 Kbyte SRAM per Transputer

Inter-Comms : Special switch chip allows full connectivity for 32 Transputers with a link speed of 80 Mbyte/s

Input/Output : RS232 external communication port

Performance : 100 MFLOPS from a 64 processor system

Operating system and langs : IDRIS, Fortran 77, C

host : IBM PC or Sun Workstation

A PARALLEL DATABASE MACHINE

Machine : Teradata DBC/1012(Teradata Corporation)
Type : Massively parallel database engine
PEs, Number, Memory : Intel 80286/386, 400 Gbyte
Inter-Comms : Ynet (2 per machine) with total 12 MHz bandwidth
Input/Output : Ethernet
Performance : 300 Transactions/s
Operating system and langs : none (not available to the user), SQL
host : Mainframes (DEC, IBM), workstations (Sun, IBM PC)

VECTOR SUPERCOMPUTERS

Machine : Cray Y-MP (Cray Research Inc.)
Type :A multiprocessor pipelined vector supercomputer
PEs, Number, Memory : powerful, 2-8, 16-128 Mword (shared)
Inter-Comms : interprocessor communication via shared memory
Input/Output : I/O subsystem with 100 Mbyte/s
Performance : 2.67 GFLOPS (8 Processors)
Operating system and langs : UNICOS, Fortran 77, C, Pascal, Lisp and Ada
host : Front-end communication with IBM, CDC etc

The type of PEs used in these machines vary from very simple to very complex ones. The simplicity/complexity of these PEs is directly related firstly, to their number, secondly, to the type of memory used in their respective machines. In shared memory machines, small number of powerful PEs are connected to a single (and large) memory store. In Alliant FX/2800, Convex C2 and others mentioned above, the number of PEs vary from 1 to 32, where each PE is extremely powerful. On the other hand, the PEs used in fine grained parallel SIMD array machines like MasPar MP-1 or CM-2 (connection machine) have a very simple structure and are mostly single bit processors. This simplicity of PEs is compensated for by a large number of them working together.

In shared memory machines, all the PEs can access the shared memory and in certain cases, as well as the shared memory, they have their own private memory. Shared memory machines are attractive because they are relatively easy to program. Most of the techniques developed for mutitasking computers, such as semaphores can be easily applied to shared memory machines. The size of memory in this type of machines is generally large and can vary from 1 to a few Gbytes (see the specification given above). One major drawback of shared memory machines is the lack of ability to scale

up. That is, as the number of PEs is increased, accessing the shared memory becomes a bottle-neck and will degrade the machine's overall speed. A solution to this problem is to add a local cache memory to each PE which would contain the commonly used data.

In contrast to shared memory machines, each PE in distributed parallel machines has its own memory. In machines such as Intel iPSC/2, or the Meiko Computing Surface, it is common for each processor to maintain its own memory and for the user program to specifically request information from another node. Accessing memory, however, in a remote location (node) will then pose a new problem of communication overheads. In BBN Butterfly, the memory is distributed amongst the PEs and are connected by a high-performance switching network. But to reduce the communication overhead, shared memory is emulated on top of the distributed memory architecture by having a global address-space and the interprocessor communications are hidden from the user.

For a large number of PEs the problem remains of how to link them together. Connecting them all to a single bus, or through a single switch, leads to the same kind of bottle-neck discussed above. Similarly, linking each PE to every other is not an acceptable solution, because the number of connections, and hence the cost, rises with the square of the number of PEs (Tabak 1990). One practical solution is to connect each PE to only some of its neighbours. One of the most popular processor interconnection topologies is the hypercube. Hypercube was pioneered by the CalTech group in the early 1980s (Fox G. 1988, and Seitz 1985) and employed in the Connection Machine, iPSC/860 and NCUBE-2. There are several advantages in using hypercube topology. First, the number of nodes in a hypercube grows exponentially with the number of connections per node, so that a small increase in the hardware at each node allows a large increase in the size of the computer. Second, the number of alternative paths between nodes increases with the size of the hypercube, which helps relieve congestion. Third, efficient algorithms are known for routing messages between processors in a hypercube. Finally, today a substantial amount of software and programming techniques exists for hypercubes (for algorithm and programming techniques see Fox 1988).

In the SM although the method adopted for the communication is based on message passing, the physical linkage between the PEs is provided by a lattice of buses. In this topology, each PE is also connected to its four immediate neighbours. Buses are fast means of data transfer and their main disadvantage is that, at any time, only one PE can access a bus. This may be a disadvantage in other architectures, but in the SM, because of the method adopted for message passing, there would be no contention on each path of the propagation (see chapter 4 and 5). In the SM, there is no restriction on scaling up as far as the shared memory is concerned but, there is a different constraint, which is

the physical limitation imposed by the number of PEs that can be attached to a bus. Each PE is assumed to have 16Kbyte of memory, which can easily cope with the applications that have high memory demand. A certain part of the memory in each PE in the SM is allocated to distributed/replicated inference engine and the communication protocols. The rest of the memory is then sufficiently used to hold a frame and all its associated information. The PEs in the SM are powerful enough to perform all the relevant operations on frames and communicate with each other and the controller. But, they are much less powerful than Intel's i860 and much more powerful than the single bit processor used in AMT (DAP) machine. Like the Teradata (see Teradata Corporation above), the type of PE that can be used in the SM can be one of the off-the-shelf ready to use PEs, like Intel's 80286. This is due to the type and the amount of information that each PE will process. In Teradata the most common operation is database transaction. Similarly, the most common operation that each PE in the SM is involved with is pattern matching. Nevertheless, each PE in the SM would cope with more computational hungry operations that may be required by the relevant applications.

The theoretical figures given above for the SM's performance are merely an estimation to enable the comparison between the SM and some of the existing parallel machines in industry. The SM's performance should be measured in the number of frames processed per second. This notion is discussed later in chapter 6, where the test runs reveal the time penalties associated with frames execution. In a parallel database machine Teradata the performance is measured according to the number of transactions per second.

4.0 CHAPTER FOUR : KNOWLEDGE REPRESENTATION AND ITS MANIPULATION IN THE SHEFFIELD MACHINE

4.1 INTRODUCTION

The computational model of the Sheffield Machine's ("SM") knowledge representation language consists of two main components : the knowledge base and the control mechanism. At the abstract level, the knowledge base describes the state of the world and the controller, by performing parallel propagation and inheritance, performs the required computation. At a more practical level, the knowledge base (more precisely, the appropriate portion of the knowledge base) is distributed amongst the rectangular array of Processing Elements ("PEs"), each frame being mapped onto a single PE. The control mechanism is also distributed and replicated. That is, every PE is capable of performing the relevant manipulation on its contents (a frame) in conjunction with the information brought in by the packet.

In chapter 2, various models of knowledge representation including associative networks were discussed. In chapter 3, some of the architectures in AI for parallel machines, including the SM, were presented. In this chapter, the SM's knowledge representation language, which includes the knowledge base and its control mechanism, are discussed. Here, the main emphasis is on the distributed/replicated control mechanism, and includes methods employed for interrogating the knowledge base where parallel propagation and parallel inheritance are involved. Note that in this chapter the discussion on the knowledge base is mainly by way of introduction, and the reader is referred to chapter 5 for full details on the SM's knowledge base.

4.2 THE KNOWLEDGE BASE

One way of organising real world knowledge is to employ associative networks (Hwang 87). In the AI community, associative networks have been explored extensively (Quillian 1976, Brachman 1978a, Fahlman 1985, McClelland 1987), and they are now widely accepted as a powerful model for knowledge representation. An associative network, as mentioned in chapter 2, consists of a collection of nodes and a set of arcs. One or more nodes denote an object, which can be a physical object, a situation, an event, or a set. Each arc represents a binary relationship between two nodes.

Despite the widespread acceptance of associative nets as a powerful knowledge representation model, the methods of encoding knowledge and labelling arcs are controversial subjects (Woods 1977, Brachman 1978a, Brachman 1983, Touretzky 1988). One can imagine the associative nets as a spectrum where, on one side,

knowledge encoding is performed using several nodes to represent an object (Fahlman 1979, Feldman 1985, McClelland 1987). On the other side of the spectrum, all the descriptions of an object and of its links with other objects, are encapsulated in a single structure (Minsky 1974, Bobrow 1985, Moldovan 1985).

It also seems that there is no established formalism by which all the possible relations amongst concepts can be defined (Brachman 1983b). Within the evaluation of semantic nets a number of links are introduced by different authors, which generally have the same implication. For example, IS-A link, is one of the most highly utilised links, in both generalised and classified hierarchies. It seems however, that this link can not cope with complex relationships that exist within real world knowledge. The other links like IS, SUPERC, AKO, SUBSET, etc not only share the same concept as the IS-A, but also the same inflexibilities (Woods 1985). In KL-ONE (Brachman 1978b) and NETL (Fahlman 1979) systems, there are sets of links which, according to their respective authors, are designed to cope with the complexities of relationships that exist between objects in the real world.

An associative network however, has expressive power to encode any object with both its explicit and implicit definition (Findler 1979). The explicit data is the object with its associated information, where the implicit knowledge is represented by the pattern in which the relationships between objects are given.

Having an explicit relationship between any two objects provides firstly, locality, which can be fully exploited in a parallel machine. Most of the related information of a given object is aggregated in a particular area in the knowledge base and relevant information can be retrieved from short distances within the search spaces. Thus, the main benefit gained by locality is faster communication with less overheads, and consequently, reduction of paging of various parts of the knowledge base.

Secondly, the network data structure contains its own embedded indexing system by means of arcs. These arcs represent the relationships between objects and are also used as a guidance for exploring the network and message passing. Thirdly, the associations amongst the frames (nodes) are extremely instrumental in disk storage, identification, retrieval and mapping operations. Fourthly, these associations are directly utilised in the hardware configuration. That is, the physical links between the PEs are set up according to the hierarchical structure of the knowledge base fragments, which is being mapped to the array of PEs.

Finally, knowledge is distributed through the network in a hierarchical form, so that operations such as inheritance of properties provide a powerful inference mechanism (Attardi 1982, Fox 1986) and data economy.

Note that data economy has a substantial influence on machines with shared memory. This is because of the limitation imposed by the size of the main memory. In contrast, in some parallel machines where each PE has its own memory of relatively modest size (eg, the SM), the emphasis would not be so much on data economy, but on the speed of processing and communication, and reducing their related overheads.

In the SM, the knowledge, after the mapping operation, is distributed in the rectangular array of PEs, where each PE contains only one frame. Within each PE, it may be possible to hold, in addition to a frame, a substantial amount of additional information. In chapter 6, the test runs and their analysis on the SM's simulation show that for further utilisation of potential parallelism, there is a need firstly, to reduce the communication overheads and secondly, for an increase in the speed of frame processing. Thus, each PE should be able to perform most of the knowledge processing itself, and has as little communication as possible. The analysis of the test runs also shows that one of the major constraints on both communication and frame processing, is the performance of inheritance where properties are inherited from higher level frames by lower level frames, in a given hierarchy.

It has been suggested that representation of real world knowledge involves a large number of tangled hierarchies (Fahlman 1987), forming a multi-domain knowledge base of a considerable size (Lenat 1987). In the SM, the knowledge base consists of a large number of heavily interlinked domains, where each domain is a combination of several tangled hierarchies that are taxonomically structured. At the higher level in each hierarchy, the generic objects contain the general characteristics of that hierarchy, which can be inherited by individual objects at lower levels. The network is assumed to enjoy all the characteristics that are known to associative nets. The local and global relationships amongst frames, across the entire network are presented by the IS-A type link. The local parent-child relationships with other frames are confined only in a single hierarchy, whereas the connections between two or more hierarchies are defined as global relationships. In the case of having more than one parent, the complexities associated with multiparentage and consequent multi-inheritance arise, in particular, in maintaining the integrity and the consistency of the knowledge base.

Storing this kind of knowledge in the SM's primary distributed memory, is almost impossible (see chapter 3, the SM's memory). That is, the number of frames in the

knowledge base would exceed the number of possible PEs (in a one-to-one mapping). Thus, because of the shortage of memory space in the SM, it is necessary to store the knowledge base on disks, as an auxiliary memory (Fox 1986), and later, retrieve the appropriate portion of it which can be mapped to the PEs. The mapping operation is on a "one-to-one" basis, which is the mapping of one frame to one PE.

In the following sections an attempt has been made to show the estimated size of a knowledge base. This involves the calculation of the size of the knowledge base, including the size of an individual node, number of nodes, and the size of the network.

4.3 SIZE ESTIMATION OF A FRAME-BASED NETWORK

We will now consider the size of a simple concept Arch represented in KL-ONE (see figure 2.4, chapter 2). There are three concept nodes, three roles, and three SD's (Structural Descriptions). The total number of links that connect these items is 30, which is probably typical. Let us assume that on average, each concept role, SD, and link is represented by a 2 byte word; this leads to an estimate of 40 words in the concept Arch, and an estimated size of 80 bytes. In figure 4.2, the frame representation in the simulation is shown. The average size of such frame can be estimated at 50 words or 100 bytes. Note, that the frames at the lower levels of the hierarchy are bound to be larger in size, and they tend to take more space than frames at higher levels.

It is assumed that the knowledge base has a network structure and contains a large number of domains, eg, the animal kingdom hierarchy can be regarded as one domain within the network, where each domain consists of one or more hierarchies. These assumptions are made to find the estimated sizes of different hierarchies within the knowledge base. Nevertheless, it is very difficult to find the exact size of a hierarchy or knowledge base before it is fully constructed. For example, in one representation of the animal kingdom hierarchy, at its second level there are 26 branches, at the third level there are 78, and at the fourth level 140 branches (Freeman 1985). At each level it is possible that a node may not have any offspring, for example at the second level there are some 10 nodes without any offspring. This highly irregular combination does not assist in the calculation of the size of the hierarchies and networks. In particular, the four levels in animal kingdom hierarchy are merely classes and sub-classes of the living things, and one can imagine the full size of the hierarchy by adding the relevant individual nodes to it. At the present time there appears to be no appropriate mechanism for measuring the sizes of hierarchies.

In calculating the size of knowledge bases, another parameter must be considered : the number of levels of each hierarchy. In the animal kingdom there are four levels (taking into consideration the living-things or the root-node being positioned at the first level), which are the combinations of classes, sub-classes and generic names of all living things. It seems reasonable to purport that in a wide range of applications, by adding the individual entities to the hierarchy, the number of levels would not exceed 10 levels. However it has been reported that the telecommunicational based system, Prestel, is based on a hierarchical structure, the number of levels being approximately 15 (Patterson 1990).
The estimated size of a tree (or a hierarchy) with ten levels and ten offspring at each level is about 1,000 million nodes. If the knowledge base contained 1,000 hierarchies whose nodes are inter-connected with great complexity, there would be 1,000 X 1,000m nodes in the knowledge base. In a frame-based system, each node may contain about eighty bytes to represent a simple concept (as estimated above), and the consequent estimated size of a frame-based knowledge base would be 80*1,000*1000m bytes. These estimations support the assumption that a knowledge base of that magnitude should be stored, at present, on disks, and that certain mechanisms should be employed to find the appropriate hierarchies and transfer them to the array of PEs (Moldovan 1986). It should be noted that the number of links connecting objects are not included in the estimation. The above calculation may sound over-estimated but the example of large size databases like the Domesday system, supports this estimation (Domesday Project 1987).

The knowledge-base is therefore a separate entity from the SM, and is stored on a set of disks. In this case, accessing the knowledge base is an important feature of the system which intends to process general real world knowledge. With the advancements in technology, it is now possible to store a large amount of data on optical disks (Fox 1986), and there are also systems that can be utilised for fast data transfer (Lavington 1987, Gray 1987).

Therefore in this system, the knowledge base is partitioned into all the domains that it contains and will be stored on disks, so that a paging mechanism will be employed to retrieve the appropriate information. When a query is made to the system, it is assumed that with the help of an intelligent-front-end (ALVEY 84), the supervisory system can be guided to fetch the appropriate domain and map it to the system. The mapping mechanism is on a one-to-one basis, so that each frame would be mapped to a PE. The mapping and paging operations will be discussed in chapter 5, whereas query decomposition and analysis will be discussed later in this section.

4.4 FRAMES STRUCTURE AND THEIR RELATIONSHIPS

In the SM's knowledge base, each node of the network is represented by a frame (see figure 4.1). Each frame contains a substantial amount of information which is partly used to define the characteristics of the object that it represents, and partly to determine its relationships with other frames in the network (eg, multiparents/parents-child relationships). In addition, certain information in each frame is used to find appropriate paths for propagation and inheritance operations.

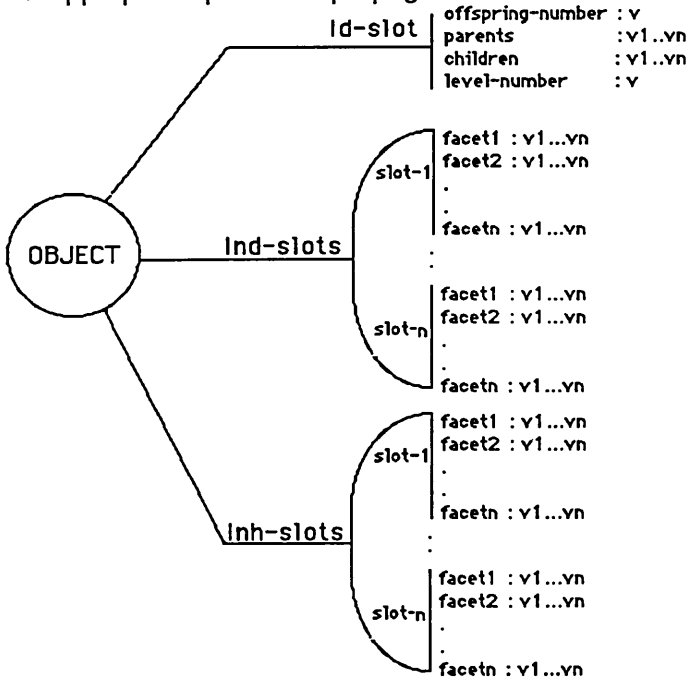


Figure 4.1, a schematic representation of a frame.

Each frame contains three types of slots :

1) id-slots
2) ind-slots
3) inh-slots.

The identification slots (id-slots), are used for data packet/block propagations/creations, frame identification, and path specification for both propagation and inheritance operation. The id-slots comprise the following facets:

a) Offspring Number : it represents the number of children of the frame. Later, this number will be used for data packet propagation.
b) Parents : it specifies the name of the parent, or names in the case of multi-parents.
c) Children : this facet specifies the children names of the object.
d) Level Number : represents the position of the object within the hierarchy.

Individual slots (ind-slots) define any individual characteristics of the object which cannot be passed down the hierarchy.

The inheritable slots of the object are given by inh-slots. These slots have been inherited from higher level parents and could be inherited by lower level frames (children) in each hierarchy. All inheritable slots have a number of facets with values, and some of these facets specify the constraints that are imposed on the slots. The inheritance operation, for example, would require specifying various constraints on how a particular property can be inherited. In general, the constraints specified by the facets ensure the integrity of the knowledge base and each individual frame. This is similar to constraints imposed on databases for entity/referential integrity (Beynon-Davies 1991).

In figure 4.2, the same data structure for a frame is given, using complex lists, which has been developed for the simulation. The first sublist contains the name of the object, and the following sublists contain the Id-slots, Ind-slots and inh-slots. In figure 4.2, there are hash numbers associated with each slot–name and the object name itself. These hash numbers are used for identification at the time of upward/downward transitive closure involving data packet propagation and inheritance operations. The hash number is an associated hashing value of a frame name, which at the time of creating the frame, will be calculated and placed at the appropriate position in the frame's structure. The mechanism for hashing and the hashing table itself, will be discussed in detail in chapter 5.

In the example given in figure 4.2, the inheritable-slots section contains four inheritable slots.

```
(((1749    a1)
    (Id_slots((offspring_no   (value   3   ))
              (level_no   (value   1))
              (parents   (value   none))
              (children   (value   ((681   a11)(781   a12)(881   a13))))))
    (Ind_slots(a1_ind_slot1   (value   a1_ind_v1))
                          (a1_ind_slot2   (value   a1_ind_v2)))
    (Inh_slots((((1749     1   )  a1_inh_slot1)
              (value   a1_v1)(inh_condition   default))
                    (((1749     2   )  a1_inh_slot2)
              (value   a1_v2)(inh_condition   default))
                    (((1749     3   )  a1_inh_slot3)
              (value   a1_v3)(inh_condition   default))
                    (((1749     4   )  a1_inh_slot4)
              (value   a1_v4)(inh_condition   default)))))
```

Figure 4.2, The implemented version of frame structure

The second slot shown in figure 4.3, is another complex list with three entries (sublists). The first entry contains the name of the slot and its associated hash number.

```
( ((1749    2 ) a1_inh_slot2)
                    (value  a1_v2)  (inh_condition  default)  )
```

Figure 4.3, representation of a slot with its facets.

The second and third entries contain two facets; value facet and inheritance constraint facet, respectively. The value facet represents the actual value of the slot, and the inheritance constraint facet represents the constraint on how the inheritance should be done.

The enormous flexibilities associated with frames allow further modifications to be done to each frame. The number of facets, for example, can be increased according to the application's requirements. If there is a need to specify the maximum and minimum number of values that a slot can have, an extra facet can be added to satisfy this requirement.

This internal definition of each object enables the control mechanism to identify the path required for the data packet propagation and inheritance operations, data packet creation, identifying the constraints on the inheritance operations, and defining the individual and general characteristics of an object.

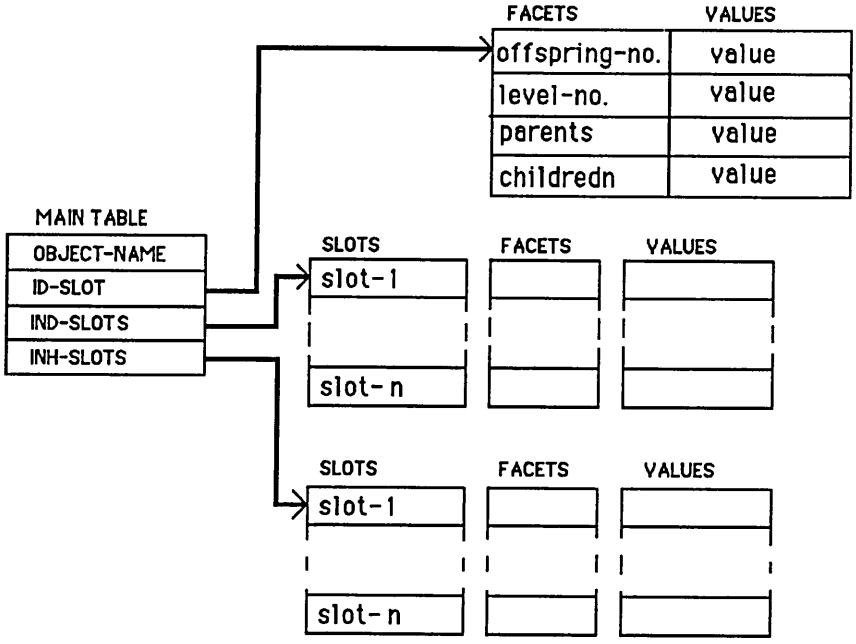


Figure 4.4, Low level frame structure.

In figure 4.4, a frame structure is shown which can be utilised at the implementation level. The low level data structure of each frame has been designed with only one point of entry, which is directly related to the nature of one-to-one mapping operation, and provides an overall consistency in respect of the path of propagation and inheritance. According to the command issued by the supervisory system, each PE can retrieve any appropriate set of slots (id-slots, ind-slots or inh-slots) via the main table. Partitioning the slots in this manner ensures correct and fast retrieval of data.

4.5 CONTROL MECHANISM and KNOWLEDGE MANIPULATION

In serial machines, the whole control mechanism is encapsulated in a single environment, and to perform any specific operations, certain instructions are serially fetched from the primary memory and executed in sequence by the CPU (Central Processing Unit). This concept, in general, is true for all the serial machines which have a single shared memory, and act in a "fetch/execute" cycle to execute all the statements defined in a programming language (Appalaraju 1984). In certain types of parallel machines, on the other hand, the control mechanism is distributed/replicated, where each PE, independent of the other PEs, can execute instructions in its own memory (see chapter 3).

In an ideal parallel knowledge base system, the overall controller should provide an environment where either a novice or an experienced user would easily be able to interact with the system. Naturally, to develop such an environment, several sub-fields of AI would be involved. The first component is a natural language understanding front-end (Moore 1986) which, as an interface between the user and the system (eg visual and/or acoustic interface), can play an important role.

The interface should be able to parse the user's input and translate it into a relevant format that is understandable by the system. According to the translated input, the control mechanism will go through an input-analysis phase as to what actions should be taken. These actions may include message passing, propagation of information and performance of inheritance operations.

In the SM, it is assumed that every PE in the rectangular array has a replicated set of instructions in its own memory, that can be invoked and executed according to the input data. As well as local control, the SM has its overall controller with the following tasks :

1 - Interaction with the outside world, which involves accepting queries and returning the integrated results back to the user.

2 - Query decomposition.
3 - Supervising the operations involved with the disk unit.
4 - retrieval of the relevant portion of the knowledge base and mapping operations.
5 - Internal communications; communicates internally with all the PEs in the array by means of broadcasting.
6 - Controls and initiates parallel propagation of data packets (message passing).
7 - Overall control over inference mechanism and knowledge manipulation.
8 - Result integration.

In the simulation, several assumptions have been made which are as follows :

1 - The knowledge base is a general purpose knowledge base, and consists of a large number of domains which are heavily interlinked. Each domain, in turn, consists of several hierarchies which implement the concepts of multiparentage and multi-inheritance.
2 - The size of the knowledge base is too large to be kept in the system's primary distributed memory. Thus, a paging mechanism is employed to retrieve appropriate portions of the knowledge base and map it to the rectangular array of PEs. Because of the heavy interlinked hierarchies and consequently multiparentage, this process may be invoked several times after the first retrieval.
3 - In the simulation, only the main characteristics of the system have been implemented, showing its response to nested queries. The tasks of complex interaction with the outside world, mapping, knowledge retrieval and other desirable features that a general purpose system would require have been implemented in a full or reduced version.

In general, the model of computation in the simulated SM can be defined as follows :

a) After the query is made, and according to the information given in the query, the appropriate hierarchies are retrieved from the knowledge base and mapped onto the machine's array of PEs. Note that, as mentioned above, while propagating and performing inheritance operations, it may be necessary for the controller to retrieve more hierarchies and map them to the array. This is because of the multiparentage feature embedded in the knowledge base.
b) A decision based upon the type of query is made as to which type of propagation and inheritance operations are required.

c) The propagation starts with the controller sending the relevant information to the appropriate PEs[1]. Each of these PEs will then create a data packet containing the query and other relevant information, which is then passed to the next level down in the hierarchy.

d) While each packet is being passed down from one PE to another PE, the inheritance operation, if required, will be performed, and its result will be stored in both the packet and the PE.

e) The parallel propagation of data packets and inheritance operations continues until the relevant conditions associated with the type of the given query are met.

In the remaining sections of this chapter, the main emphasis will be on the inference mechanism developed for the simulation of the SM. This involves the methods of propagation; parallel message passing and parallel performance of inheritance operations and a discussion of the query-system developed for the simulation.

4.6 INFERENCE IN THE SM

The inference facility in associative networks, particularly in frame-based networks, can be rich, flexible and powerful (Ffuruhawa 1981). The source of this versatility comes from the special characteristics of frames and their associations within the network. The main inference mechanism in a frame-based network is based on the relationships amongst frames, and on the inheritance of properties. At the higher levels, in each hierarchy, generic concepts define the general properties which are passed down to the frames at the lower level. Through these transactions, inferences are made. Further inferences can also be made that are based on the hierarchical relationships amongst the frames.

A frame in the network can contain different types of information, including procedural and declarative knowledge (Winograd 1985). Rules, for example, by referring to the frame's slots, can be used to reason about certain characteristics of that frame. If an application so requires, one can increase the depth of inference in the system by employing the rich facilities that frames offer. In most of today's AI tools, frames are used as the basic representation scheme (IntelliCorp 1986, Clayton 1987, Laurent 1988).

---

[1] Here, the appropriate PEs are those which contain the injection points that can be root nodes or any other frame at higher levels of every relevant mapped hierarchy.

The inheritance of properties and consequent inferences rely quite heavily on matching operations. A small portion of the knowledge structure is constructed according to the external query, and matched against the knowledge base to see if such an object exists (Barr 1981). During this process the control mechanism may deduce that the new knowledge structure should be added to the knowledge base. The Sniffer system, by Fikes and Handrix, is an example of a network deduction system based on matching (Ringland 1989). Nevertheless, in certain types of associative networks, rather than just relying on matching as the basic operation for inheritance, additional methods have been employed. Spreading activation in Quillian's TLC (Quillian 1968), and VC links in NETL (Fahlman 1979) are two examples of such methods (see chapter 2 & 3 for TLC and NETL respectively).

These extra mechanisms have emerged along with the new developments involving parallel machines, where in some cases, the knowledge and control are distributed (or replicated). Some of these machines, like SNAP (Dixit 1984) and the SM, have distributed memory which houses the knowledge according to its predefined structure (eg, one frame/PE). The inheritance of properties and consequent inferences in such machines will, therefore, tend to be different to that of serial machines. In serial machines, the execution takes place in sequence and there is considerable restriction and complexity imposed on the system by the shared information. In parallel machines, on the other hand, the emphasis is shifted onto reducing the communication overheads by avoiding shared information, finding appropriate methods for propagation, message passing, inheritance and inference making.

In the following subsection, there is a discussion of a new approach implemented in the SM system which involves parallel propagation of data packets, parallel performance of inheritance of properties and parallel matching operations.

4.61 PARALLEL PROPAGATION OF MESSAGES

In the SM system, the knowledge base is hierarchically structured so that the properties of generic objects at higher level of a hierarchy can be passed down to more individual objects at lower level. The mechanism for passing such knowledge from a frame to another frame in each hierarchy, is known as inheritance. A frame in the network can inherit a slot, slot-values, procedures, inheritance constraint and null (ie no inheritance). In the case of multiple inheritance, a frame receives properties from more than one parent, which can be anywhere in the knowledge base. Local multiparentage involves parents in the same hierarchy; whereas a global parent is a frame from another hierarchy.

As a result of performing inheritance in the SM, inferences are made. In order to be able to perform inheritance operations, and consequently, inferences, relevant information should be propagated through the SM's array of PEs. Any method that is to be developed for propagation must bear all the complexities involving the above operations, and at the same time, must be as efficient as it can be. In a system communication of this type, the most essential element is the speed of communication, despite the size of messages and their asynchronous behaviour. The information and messages are continuously updated and accumulated as the result of inheritance operations.

In the test run analysis of the SM, in chapter 6, it is shown that one of the most crucial aspects of parallel processing is to reduce the communication overheads. The communications in the SM system involve the controller and all the relevant PEs in the array. The controller broadcasts information to PEs via a set of buses, and the PEs communicate with each other and the controller, via the same media (see chapter 3). This is an important part of the system, where all the potential parallelism could be fully exploited, otherwise the system will fail to be cost effective.

The propagation of messages involves data packets which contain information that are initially sent by the controller, and then passed from one frame to another within the array of PEs. While these packets are being propagated, as a result of matching and inheritance operations, their contents are updated. Thus in the SM, data packets provide two important services : communication and knowledge manipulation.

Knowledge manipulation is the term used to describe tasks such as pattern-directed searching, and adding to, deleting or modifying the knowledge-base (Gray 1987). Pattern-directed searching, in our model, is based on propagation of data packets through the hierarchical relationships (paths). In each hierarchy, each frame is associated with its ancestors and descendants, except the root-node which is connected only to its descendants. These associations, or in this case, paths, are the directions that the propagation will use. In the network of PEs, while data packets are propagating, at each node (PE) the relevant contents of the incoming data packet are matched against the appropriate parts of the frame residing at that node.

After the controller -according to the given query- has identified, retrieved and mapped all the appropriate hierarchies to the array of PEs, it initiates the propagation by sending the query-frame[2] to the injection points. The injection points are those PEs which contain the frames representing the root-nodes of all the relevant

---

2 The query-frame is a data structure similar to a frame that contains the information given by the user.

hierarchies. Each of these PEs creates a data packet containing the query-frame and other relevant data that they have received from the controller. In each hierarchy, the data packet starts moving down from the root-node and will be replicated at each frame with children, one copy being sent to each child. Figure 4.5 shows the propagation of messages starting from the root-nodes.

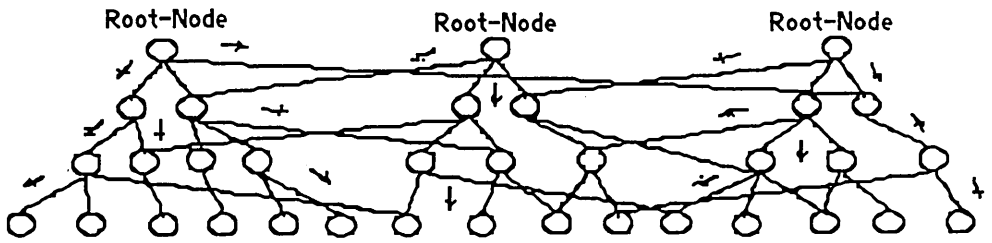


Figure 4.5 Data packet propagation.

This operation continues in parallel at each branch in each hierarchy, until the required conditions are met.

This method of propagation utilises the potential parallelism in the system, particularly where the hierarchies are bushy and the number of levels is small. As the inheritance and matching operations are performed from the higher level to the lower level of all the hierarchies, the whole process of finding a solution to a given query will necessarily go through a number of cycles. In contrast to serial machines, the process of exploring a large volume of data can be reduced to a few cycles, which are the same in number as the maximum number of levels in a given domain. It is clear that all the hierarchies in a knowledge base representing real world information will not have the same number of levels, but it has been suggested that the maximum number of levels would not exceed ten (Fahlman 1985, Walters 1988).

Nevertheless, the communication overheads caused by data packets moving down hierarchies, should be carefully monitored. The size of each packet and the consequent time penalty caused by transferring these packets down the hierarchies are important issues. In the simulation, with the aid of upward pointers, the inheritance operation will be performed only on appropriate frames and their slots. The result of this optimisation is that the size of data packets is reduced, which ultimately will result in a reduction of transfer time.

Let us summarise the main characteristics and activities involved in this type of propagation. The propagation begins by data packets moving down from the root-nodes (or injection points) of all the hierarchies that have been mapped to the array. Each packet contains the query-frame and other necessary information and while moving

down, as the result of matching and inheritance operations, information can be deleted from, or deposited in, the packet. Thus, a particular frame may inherit some values for its slots and/or at the same time it may have some information which can be inherited by lower level frames. All the packets at each stage of propagation will contain the most updated data and, before moving down to the next level, their contents will be copied and stored in the PE that contains the relevant frame. Thus, each packet at any stage of the propagation (or cycle), contains all the inherited information from higher levels and each PE has a copy of it. This information can later be used by the controller or even by the user if any queries are made concerning that frame. Note that these operations are performed in parallel, level by level in each hierarchy, involving all the relevant PEs in the array.

The strategy used in this type of propagation can, in general, accommodate all types of queries. All the queries developed for the SM utilise this method of propagation, which offers the flexibility required for a general system. In this system, the structure and the functions of the controller are made simpler, and the emphasis and the responsibilities are shifted onto the array and all the PEs involved. In a parallel environment, this is an important issue, where the task of resolving a problem is given to a parallel distributed system rather than to a single control mechanism.

4.62 PARALLEL INHERITANCE OF PROPERTIES

It was mentioned that the propagation in the SM starts after the controller has located, retrieved, and mapped all the hierarchies relevant to the given query. The controller, by checking slots of the query-frame against its original version from the knowledge base, determines the injection points in all the hierarchies[3]. The injection points are the frames residing at the highest points in each hierarchy, from which the query-frame is inheriting properties. The parallel propagation starts by the controller broadcasting the query-frame to every PE that contains an injection point. Each of these PEs then creates a data packet which includes the query-frame, and after performing initial inheritance and matching operations, sends it down the hierarchy. While each data packet is moving down, at each level on each frame, the inheritance of properties and matching operations are performed.

Each inheritable slot in a frame in the knowledge base has two facets. The first facet represents the value of that slot and the second facet defines the inheritance constraint on how the value should be inherited. There are four different types of inheritance

---

3 In some of the queries available in the SM the frame-name may be absent. The controller then starts the propagation from the root-nodes of all the mapped hierarchies.

constraints; default, union, intersection and override. The default constraint allows the inheritance of slot/values. The union constraint requires the inherited slot/values from one or more parents to be combined with the existing ones (without any repetitions). The intersection constraint will allow only the common elements of inherited properties from more than one parent to be inherited. The override constraint allows null inheritance, which provides exceptions that add to the flexibility required for representing real world knowledge.

There are different methods for how and when the inheritance operations should be performed. The full inheritance operation can be performed while the knowledge base is being created. Another option is to perform the inheritance at run time (or query time). This option will mainly involve default reasoning, because if there are going to be any exceptions (eg override), the user has to intervene to modify the default values while querying the system. The third option is to perform part of the inheritance at assert time and the rest of it at run time.

In the SM, the ADT[4] (Application Development Tool) is used by the user to create a knowledge base. For a newly created frame, all the slots from higher level frames will be automatically inherited on a default basis. The user can then make the appropriate modifications as to what slots and under what inheritance constraints they should be inherited. Further, the user can add some inheritable slots to the frame which will be inherited at lower levels of the hierarchy. Thus, at the time of knowledge base creation (or assert time) in the SM, a semi-inheritance operation is performed by the user's interaction. In this operation each new frame will inherit only those slot-names that the user has decided are useful, and the values of those slots will be inherited at run time, when the user makes a query.

The mechanism used for the inheritance operations in the ADT is exactly the same as that used for inheritance operations at run time. Also, in both cases (knowledge base creation and query time), all the relevant portions of the knowledge base are present in the system or will be mapped as soon as they are required.

Another complexity involved in parallel inheritance operations is, how should frames at lower level inherit slots and their values from higher level frames ? And, how does the distributed/replicated control residing in each PE determine the coupling of appropriate slots and their values ?

---

4 The operations of the ADT are discussed in chapter 5.

In figure 4.6 for example, the frame "Ian" should inherit all the properties from "Human", "Adult-Male" and "Married-Adult-Male". This concept can be made more complicated by assuming that the frame "Ian" inherits properties from hierarchies other than its original one. This is also shown in figure 4.6, where "Ian" has a car of type BMW 525e and plays sports of type volleyball.

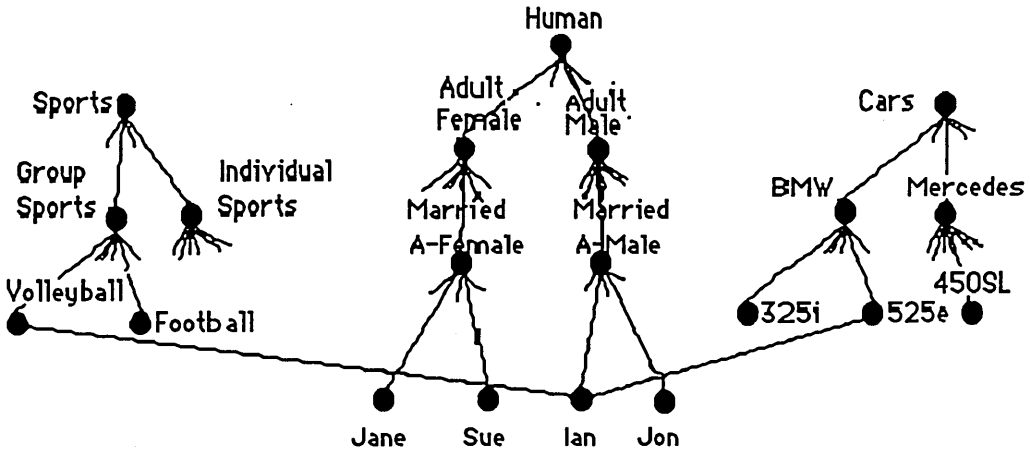


Figure 4.6, fragments of three different hierarchies.

There are two alternatives to determine how a frame can inherit its appropriate slots and their values. The first alternative is to perform inheritance by default; ie, frames at lower level will inherit every properties from its ancestors. In the second alternative, the system allows each frame to inherit only specified properties.

The first alternative imposes simplicity as to how the inheritance operation should be performed. That is, the only inheritance constraint that can be implemented is the default inheritance, which in representing real world knowledge may cause severe limitations. Having default inheritance only, will allow the system to be simple and limited where an inheritance operation will be performed for every slot that a frame can inherit. There would therefore be no need to assign hashing values for each frame and its inheritable slots (see figure 4.7), to perform the coupling operation. The major limitation in this method is that exceptions within individual frames cannot be represented. In the example above, "Ian" may inherit all the properties of being a sportsman by playing volleyball, having a BMW but by being divorced will threaten the consistency and integrity of the knowledge base. In such a knowledge base, there is no provision made for dealing with the exception, which in this case is being divorced. The default assumption is that whoever the person is, as long as he is an adult married male, he will have a wife with some name. Thus, the simplicity provided by default reasoning creates uncompromising problems for the knowledge base including inconsistency and incorrect representation.

In contrast, the second alternative provides better flexibility and more facilities, while adding to the complexities of the system. In this alternative, there are hashing values that are used to provide correct coupling between a slot and its value. The advantages and disadvantages of this kind of implementation are as follows :

- Advantages :

  Correct and easy coupling between a slot and its value.

  Ability to include different constraints on what to inherit.

  Ability to provide exception whenever is needed.

  Ability to increase the efficiency of the system by optimising the amount of inheritance operations performed in each PE.

- Disadvantages :

  Extra complexity, resulting from the development of hashing algorithm.

  Embodying the hashing values in each frame in the knowledge base.

  Increase in size of each frame in the knowledge base.

  The danger of being forced to utilise shared information.

By examining these two methods of performing inheritance operations, it seems the most suitable method is that of the second alternative. That is, there should be some additional information in each frame which enables the inheritance mechanism to effectively find appropriate slots and their matching values. This contradicts in part with what was said in chapter 2, by claiming that inheritance operations provide data economy. But, since we are dealing with a parallel machine with distributed memory where each PE has a large enough storage to hold a frame with its full additional data, the contradiction can be ignored.

In the SM system, frames are regarded as the building blocks of the knowledge base. Each slot in a frame contains the slot-name and two facets to define the inheritance constraint, and the value of the slot, respectively. Associated with each slot-name there is a hashing value of a frame from which that slot has been inherited. In figure 4.7 an example of a frame is given.

```
(((H-No2   Adult-Male)
  (id_slots ((offspring_no (value xx2 ))
             (level_no (value xx2))
             (parents (value (H-No1 Human)))
        (children (value ((H-No5   Married-A-Male).....)))))
   (ind_slots   (ind_slot1 (value ind_v1))
             (ind_slot2 (value ind_v2)))
   (inh_slots   ((((H-No1   1 ) date-of-birth) (value nil) (inh_condition default))
```

(((H-No1 2) place-of-birth) (value nil) (inh_condition default))

(((H-No1 3) no-of-children) (value nil) (inh_condition default))

(((H-No1 4) occupation) (value nil) (inh_condition default))

(((H-No2 1) Wife-name) (value nil) (inh_condition default))))

Figure 4.7, an example of a frame in the knowledge base.

In figure 4.7, a frame is shown with all its three groups of slots. In id-slots, the frame's parents and children names with their associated hashing values are given. In this frame there are additionally 5 slots, four of which have been inherited from its parent : Human. The last inheritable slot is to be inherited by Adult-Male's descendants. All the inheritable slots, whether they are inherited from higher level frames, or are to be inherited at lower levels, have associated with them a hashing value of the frame of their origin. The frame in figure 4.7 is a generic frame, which means its slots are general properties for its descendants. Thus, those slots usually have nil-values and frames at lower levels will have individual values for these slots.

Another important issue is whether the value inheritance should be from the slot in the original frame, or from a frame at lower level, where some modification has been made to it (Brown 1988). In other words, should the pointer (hashing value) point to the frame where the original slot is being inherited from, or should it be pointing at a frame incurring the most recent modification ? This is an important issue, since these points will be used to inject data packets, to start the propagation and perform inheritance and matching operations.

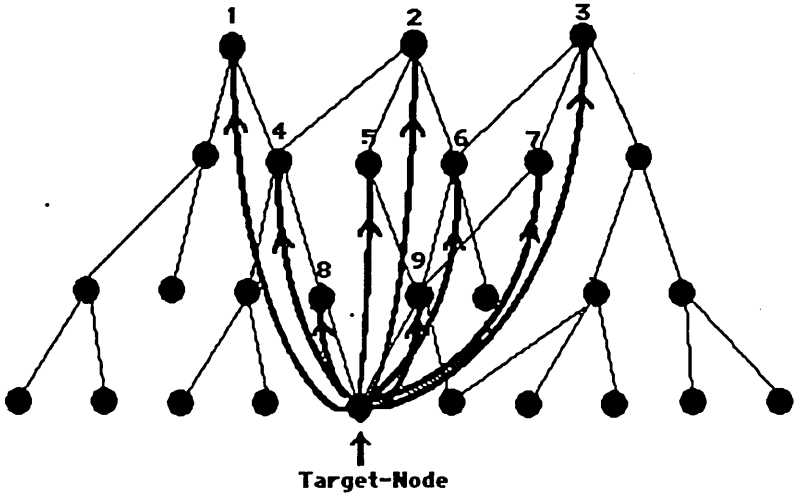


Figure 4.8, an example of upward pointers.

In figure 4.8, a target-node (query-frame) is shown in a small fragment of the knowledge base. The hashing values (or upward pointers) associated with each slot shown in figure 4.7, are represented by upward arrows in figure 4.8. These arrows are pointing at frames from which the slots for the target-node have been inherited.

The injection points, as mentioned above, are important because, firstly, they represent the starting point for the propagation, and secondly, they affect the way that the propagation of messages is conducted.

Alternatively, the propagation can start from the root-node of each relevant hierarchy ignoring the injection points.

It was mentioned that the propagation will go through a number of cycles, where in each cycle, relevant inheritance and matching operations will be performed. The completion of all the cycles may be crucial, so that the requirements set up by the query can be achieved. Thus, the time taken for the whole propagation will be as long as the time taken for the longest cycle. Therefore, whether we choose the first or the second option, it may be possible that one of the injection points is the actual root-node of one of the relevant hierarchies. This means that, in spite of early completion of other cycles, for the propagation to end successfully we must wait for the longest cycle. If this is the case, then the time and effort spent on determining the injection points, and on other relevant computations, has been wasted.

In figure 4.8, for example, if the controller initiated the propagation by injecting packets at nodes numbered 1,2,3,4,5,6,7,8 and 9, the target-node will receive packets from nodes 8 and 9 much earlier than, say, node 1 and it will still have to wait for the packets from other nodes in order to make any conclusions.

4.7 QUERIES

Queries are the means of providing information retrieval through interaction between two different entities : human to human, human to machine etc. In natural languages for example, queries are complex and mostly implicit, and the difficulties lie in decomposing them into their most elementary form (Schank 1972, Woods 1978). In computerised systems on the other hand, the queries tend to be simple (Deen 1985). Nevertheless, there are query languages developed for databases and expert systems providing a wide range of different queries (Date 1984, Forsyth 1985). In database systems queries are used for information retrieval[5], whereas in expert systems, as a result of querying the system, certain modifications including additions and deletions can be made to the knowledge base (Hays-Roth 1983, Jackson 1990). This is equivalent to the DML (Data Manipulation Language) used in database systems (Date 1986).

---

5 Most of the query languages used in today's databases, as well as data retrieval, provide added facilities for data-manipulations.

In knowledge base/database systems, a query is a statement requesting the retrieval of information from a bank of data- databases or knowledge bases (Frost 1986). Broadly speaking, there are two types of queries : procedural and non-procedural (or declarative). Procedural queries involve the specification of what data is required and how it should be retrieved; whereas in non-procedural queries, it is sufficient only to ask for what data is needed and the system itself will perform the retrieval operations (Benoit 1986).

In sophisticated knowledge base/database systems highly advanced graphical environments have been developed to provide better interactions between users and the system (KEE, ART etc). Facilities are also provided for the translation and decomposition of the user-queries into forms specific to the internal structure of the system. In large knowledge base/database systems, as the knowledge/data is stored on disks, an important role is given to different mechanisms for identification and retrieval of relevant information from the appropriate knowledge base (or database).

Applications involving knowledge base systems include interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, control and many others (Hayes-Roth et al. 1983), most of which engage in a dialogue with the user. The system may suggest options based both on the knowledge that it possesses and the data given by the user. All the interactions can be either user initiated or computer initiated (Jonson 1988). MYCIN for example, interacts with the user by starting with a general set task like "compile the best therapy regime for this patient". Following the initial action, the system requests input that will enable it to make the initial inferences. Further inferences can be made by further requests for input until the task is accomplished (Clancey 1984). In the user initiated mode, computer systems are restricted only to the user's request. The accuracy of any conclusion or recommendation reached by the system is influenced by the amount of data provided by the user, eg programs in a high level language like Pascal or Lisp.

4.71 DIFFERENT QUERY LANGUAGES

Today's AI based systems or database systems, are still limited to a small set of queries. These queries can be made through very high level languages, which describe retrieval operations and data manipulation involving insertion, deletion and modification. These facilities are embedded in languages like SQL : Structured Query Language, QBE : Query By Example, (Date 1986), TellAndAsk (IntelliCorp 1986), rule-based manipulation in ART (Clayton 1987).

In database systems, the software is built on an underlying data model. There are three fundamental database models : the Hierarchical model, the network model (CODASYL) and the relational model. In the hierarchical model data is organised hierarchically in relationships of ownership. In the CODASYL model, the concept of hierarchy is extended into the the concept of a network. Data redundancy and detailed navigational problems are two major set-backs with the hierarchical and network models respectively (Date 1986). In contrast to both these models, the relational model has a sound theoretical basis and thus has gained some supremacy.

In both network and hierarchical databases, the retrieval and update languages (DML) tend to be procedural. In other words, the propagation is explicitly navigated by the user's program rather than by stating the properties of the data of interest (declaratively). In a declarative system, such as a relational database, the Data Manipulation Language is a derivative of relational algebra or relational calculus. In these languages the operations are specified in terms of names and values only.

One of the most utilised query languages in database systems is SQL, which is fundamentally a query language based on the relational calculus. It is a declarative query language in contrast to procedural query languages based on relational algebra. In other words, in SQL the user will only need to specify the problem and the method of how to solve it will be left to the underlying layers of the system. It should be mentioned that SQL is not simply a query language, and it is becoming the standard interface to relational and non-relational database management systems (Beynon-Davice 1991). There are three major parts in SQL : DDL (Data Definition Language), DML (Data Manipulation Language) and DCL (Data Control Language). These components of the SQL are used for data/table creation, data insertion, deletion and integrity maintenance of tables and their components, and data retrieval.

Although SQL has a data definition and data control facilities, it was designed primarily for data retrieval. The extraction is accomplished by combining select, project and join operators of the relational algebra. Simple selection can be performed as follows :

SELECT <attribute-1 name>, <attribute-2 name>, ..., <attribute-n name>
FROM <table name>
WHERE <condition>

In SQL, the structure has the ability to nest queries in select statements. For instance, to find out who earns more than Smithy, we could write :

SELECT employee-no, name
FROM employee

```
WHERE   salary >   (select   salary
                   FROM employee
                   WHERE name  = 'Smithy')
```

In the nested queries in SQL, the innermost query is evaluated first .

The join operations in SQL are performed by indicating common attributes in the where clause of a select statement.

```
SELECT   salesmen-no, salesmen-name, customer-no, customer-name
FROM     salesforce, customers
WHERE    salesforce.sales-area = customers.sales-area
  AND    customers.sales-area = 'Sheffield'
```

In the example above, the select statement extracts data from the salesforce and customers tables of relevance to salesmen working in the Sheffield sales area.

In AI based systems, mostly knowledge based systems, because of the small size and narrowly defined application domains, the queries tend to be specific and limited[6]. But, it is now widely accepted that for a knowledge base system to be both commercially and scientifically effective, it should possess a large amount of knowledge. This type of knowledge base system would also require computational properties that the previous systems could not deliver. Fahlman suggests six computational operations which he believes any large knowledge base system should support (Fahlman 1987). These operations define the computational ability of an intelligent system, which can provide facilities to develop complex queries.

The computational operations are as follows :

a) Set intersection.
b) Transitive closure.
c) Context and partitions.
d) Best-match recognition.
e) Gestalt recognition.
f) Recognition under transformation.

These computational opertations are, in theory, suitable for parallel machines, which should speed up their execution. To do this, the machine must be able to cope with a large search space and be armed with a powerful reasoning mechanism.

---

6 The reason for this is the lack of power and resources (see chapter 2 for more detail).

Lavington suggested a set of operations for associative nets which include both knowledge manipulation and knowledge retrieval (Lavington 1987). These operations include insertion, deletion, pattern-directed search, path traversal and upward/downward transitive closure, and are implicitly or explicitly related to the queries available in a system. A query, for example, may require the recognition and retrieval of certain information which involves pattern-directed matching and transitive closure.

In the KEE system (see chapter 2), there is a query language called TellAndAsk, which provides essentially three basic functions : ASSERT to create a fact, RETRACT to remove it, and QUERY to extract knowledge from the knowledge base. The visual interface provided by the KEE system offers two different types of formats for knowledge manipulation in TellAndAsk : an English-like form and a prefixed form.

KEE's TellAndAsk is similar to SQL. By using TellAndAsk, the user is able to add new information to the knowledge base, retrieve information from the knowledge base and query the information in the knowledge base. Similar to SQL, TellAndAsk has its own vocabulary, grammar, functions and operators. As mentioned above, there are three principle operators in TellAndAsk; assert, retrieve and query. Each operator is capable of performing several operations. The query operator enables the user to ask about units, slots, slot values, value class specifications, subclass and member relationships, unstructured facts and it initiates backward chaining when a rule class is specified. There is a facility in the TellAndAsk called "Ask.User", which is used to supply new information by engaging in dialogue with the user.

The query operator provides answers to the questions given about facts in the knowledge base. The operations involved in 'query' are based on matching a pattern provided by the user and the facts from the knowledge base that have been retrieved. The query pattern may contain variables the value of which is determined by corresponding facts retrieved from the knowledge base. Variables are used in slot or frame positions. A simplistic query in KEE using this format is given below :

(QUERY '(ALL ?Z ARE MAMMALS) NIL NIL : HOW MANY 'ALL)

In this query, the user has asked for all the subclasses of mammal. The 'query' would return a list of all the subclasses of mammal and 'nil' if there is no subclass for mammal in the knowledge base. In general, the operator 'query' will return a list of instances of the query pattern otherwise it will return nil.

TellAndAsk provides several other operators which can be combined (nested) under the rules of syntax provided by its grammar to form more complex queries. The compound expressions can be linked by logical connectives AND, OR, NOT.

In ART the facts drive the rules (data driven computation). Facts are matched against the rules in the knowledge base until the pattern(s) in a rule matches a fact (Clayton 1987). Similar to many rule-based systems like Prolog, ART backtracks to instantiate the remaining unmatched patterns. When all the patterns of a rule have been instantiated, it will be sent to the 'agenda', which is a list of all the triggered rules currently competing for an opportunity to be fired (activated). ART chooses the most suitable (or important) rule and executes it. After firing a rule, ART revises the agenda, taking into account any changes made in the database, and executes the next rule. This cycle repeats until given conditions are met, or the agenda is empty. The whole process is similar to that of rule based systems, production systems or Prolog.

In ART rules are used to manipulate schemata (frames) and facts. The left hand side of each rule consists of conjunctive/disjunctive clauses (conditions, or patterns). After all the conditions of the left hand side of a rule are met, it would be placed on the agenda. Conditions on the left hand side of each rule form patterns which may contain variables.

To make queries in ART, one of the following formats can be chosen :

a) writing patterns about facts

```
(defrule  find-colour
   (Colour ?x   &   ~red)
; find a fact that contains ?x with a colour that is not red
=>
(some actions)  )
```

b) writing patterns about schemata (frames)

The pattern mimics the format of compiled frame (this is called query-frame in the SM). A frame representing the generic concept 'grass' for example is as follows :

```
(defschema grass
(is-plant)
(seed-type monocotyledon)
(veins parallel)  )
```

To make a query to find the name and stem type of all monocot plants, a pattern would be created as follows :

```
(defrule  find-parallel
   (schema  ?plant)
   (seed-type monocotyledon) ; must be a monocot
   (stem-type ?stem)  ; what kind of stem. This is inherited from higher level frames
=>
(some actions)  )
```

The patterns (or conditions) are either conjunctive (ANDed) or disjunctive (ORed). The overall view of querying frames with OR and AND is :

```
(defrule  <name>
(OR    (AND  (pattern-1)
             (pattern-2)  )
       (AND  (pattern-3
             (pattern-4) )   )
=>
(perform some actions)  )
```

In ART facts and frames (schemata) can be asserted, deleted and modified. These operations are performed by defined rules with their right hand side containing the information to be manipulated. For the information to be added, deleted or modified, the right hand side would contain the combination of relevant operators such as assert, retrieve or retract. The information provided by these operators are held in variables. If, for example, a frame is to be added to the schemata-base, we can use the existing information in the facts-base to create the new frame :

```
(defrule  assert-car-schema
   ?x  <-  (car-colur  ?colour)
   ?y  <-  (car-owner  ?person)
   ?z  <-  (car-cost  ?money)
=>
   (retract  ?x  ?y  ?z)
   (assert (schema car
                (colour  ?colour)
                (owner  ?person)
                (cost   ?money) )))
```

This rule will bind the variables to appropriate values by matching operations in the facts-base and then create the frame using these values.

To impose control on the flow of execution, the rules are ranked in terms of importance. That is, each rule will have a certain weight attached to it and, when necessary, the system will fire a rule which has the largest weight.

Note that in ART, facts and frames after being compiled are all presented in the same format. The compiled facts are presented in the form of :
f-1107 (colour fifi black).

This format is the same for schemata where each slot is represented as a fact with an id-number. This implies that the ART control system can access the facts-base and schemata-base in the same manner. The pattern matching rules that govern the query system in KEE follows the same philosophy. This indicates the fact that all the information in the knowledge base (or fact-base) in both KEE and ART are kept as n-tuple entries with id-numbers (ie, a hashing values) used for fast identification and retrieval.

4.72 QUERIES IN THE SM

A query language has been developed to interrogate the SM's knowledge base. Through each interrogation, attempts are made to study and examine the behaviour of the SM, which is characterised by parallel communication and knowledge manipulation. The examination involves testing the system and its parallel environment for its suitability/feasibility and for possible gain in speed of execution.

A query language, however, should utilise all the facilities and properties of the adopted model of knowledge representation. In relational databases, operators such as 'select', 'project' or 'join' fully exploit the characteristics of the underlying data model. In a production system, such as Mycin, the application domain is narrowly defined. The scope of its query language is limited to the patient, cultures and identified organism in them, in order to suggest a suitable drug (Shortliffe 1976). Further, a query language should enable its users to define functions for further manipulation and retrieval. These user-tailored functions would be embedded in complex queries in order to simplify their appearance to the familiar process of procedural abstraction.
A query language can either be completely independent of the system's execution or fully dependent on it. The major advantage of an independent query language is its mobility, with a possible disadvantage in increasing computational cost. In the SM, a

query language has been developed which is superimposed on the underlying model of computation, that is the propagation of the messages in the manner of cyclic downward transitive closure.

4.721 TYPES OF QUERIES IN THE SM's FRAME-BASED LANGUAGE

In the SM's frame-based system, classes and their instances are clustered together hierarchically; where classes define the structure, characteristics, properties and the behaviour of their instances. In this approach, all the units have a uniform structure, in which slots represent the properties and characteristics, and procedural attachments describe their behaviour.

The attribute (slot) definitions or properties of generic objects (or classes), but not their values, are shared by their instances. In other words, a generic object has "n" number of slot-names most of which have no values. The values for these slots, at lower levels, will be instantiated according to certain constraints. Notice that the equivalent declarative definition of slots in a procedural world may be referred to as "variables". In this light, it may be possible to distinguish between class variables and instance variables, which are class-slots and instance-slots respectively.

In procedural programming languages, like Pascal, a variable can have its value changed according to the program's instructions (multiple value binding). In Prolog a variable can have only a single value. That is, when a procedure is called, its formal arguments will be instantiated to the actual arguments at the time of calling the procedure (single value binding). To evaluate an expression in Lisp, the atoms in the formal parameter list of a function are bound to the values of the actual arguments. Lisp then retrieves the body associated to the function name and evaluates it. It can be seen that evaluating a function is like evaluating its body of code, with the formal parameters replaced by the values of the corresponding actual arguments. This is called zero value binding.

In ART, in its pattern matching operations, the variable is bound to a value only for the duration of the rule (ie, holding the value within one activation). This applies also in KEE , where the variable binding is made during each execution.

Knowledge processing in the SM involves pattern matching, where frame-names, slot-names and slot-values can have their own respective values, or in their absence, variables to represent them. In the same process, as a result of inheritance operations, appropriate values replace variables. But these variables do not have the same meanings as those of shared memory machines. In the SM, each PE processes the

incoming data in accordance with the information that it contains in its own memory. The data is contained in a datapacket, which is a data structure containing frame-names, slot-names and slot-values that are either being inherited from higher level frames, or are provided by the user.

In such an environment, two main approaches are adopted for querying this frame-based model : frame-related conjunctive queries, and domain-related conjunctive queries.

In the first method, queries are made with various conjunctive components of a particular frame, where the smallest unit is a slot. Below is an example of this type of query :

```
((hierarchy's  name)
  (?frame-name
  (?slot  :  2)
  (Married  :  ?value)
  (car  :  BMW)
  (plays  :  volleyball)   )
```

Find **?frame-name** who has **?slot** = 2, and married with **?value** and drives BMW and plays volleyball.

Most of the conjunctive clauses in this type of query are properties that are inherited from other hierarchies and domains. This is similar to that of ART and KEE, where slots are represented as separate facts (not part of any frame). The only difference is that in the SM there is no separate fact-base (or slot-base) and all the information is encapsulated in frames.

A variety of queries can be made with this approach. The simplest version is to ask for the full details of a frame, where the user is required to provide only the frame-name. Another version is to provide a complete, or partial detail of a frame in order to find its name. In such a case the user has to provide the names of appropriate hierarchies. In the example above, while creating the query-frame, the user would be asked to provide the names for the relevant hierarchies. In appendix D some of the test runs for this type of query are presented.

The emboldened words in the example given above represent variables that after appropriate pattern matching and inheritance operation will be replaced by values. The result may be as follows :

Mr Smithy has 2 children and is married to Linda and drives a BMW and plays volleyball.

In the second method, domain-related conjunctive queries, a variety of queries can be made. A query related to subclass relationships amongst frames in one or more hierarchies can be made, where the user can ask for the relationships between a given frame and its ancestors/descendants.

Another query in this approach is to make a complex query, where each conjunctive clause is shared by a group of frames. The example given below justifies this concept (see figure 4.6) :

(?frame-names
  (drive BMW, 525e)
  (play volleyball)
  (Occupation lecturer)
  (salary > £40k))

In this query, all the clauses are conjunctive implicitly (syntactically speaking) and it is asking for the names of all those people who drive BMW of type 525e, play volleyball, work as lecturers and earn more than £40k.

Another example of this type of query is to ask for shared information amongst different hierarchies shown in figure 4.9.

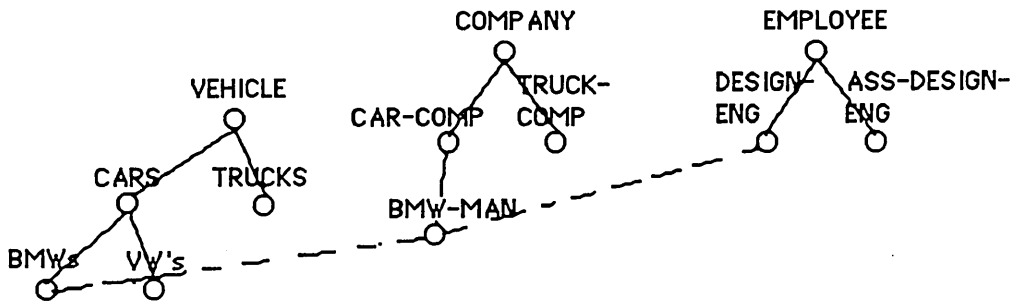


Figure 4.9, an example of a different hierarchies.

Figure 4.9 represents graphically (without their internal data) a small portion of each of three different hierarchies : vehicle, company and employee. The details of each root-node for the above hierarchies is as follows :

```
(vehicle
  (Colour : nil)
  (engine-size : nil)
  (manufacturer : nil)  )
(company
  (company-name : nil)
  (location : nil)
  (design-engineer : nil)  )

(employee
  (empl-no : nil)
  (empl-name : nil)
  (age : nil)  )
```

We can ask for all the red cars manufactured by a company located in Stuttgart whose design engineer is under 50 years of age.

In a relational database system (see section 4.73) the three hierarchies would be represented as three relations. To make a query like that above, it would have to be formulated as a join of these relations. In the SM's frame-based knowledge base on the other hand, the frame "BMW's" inherits slots/slot-values from its link to "BMW-Manufacturer" frame. The offspring of "BMW's" would also inherit this information and would be identified in the propagation. In the resulting integration process, all the offspring of "BMW's" that meet the conditions set in the above query would be returned as the answer to the query. In appendix D some of the test runs for this type of query are presented.

Note that, in ART and KEE, frames are stored as groups of facts which means both facts and frames are stored in the same format. It is only with tokens or identifiers that frames can be distinguished from facts. It seems this approach is directly linked with the storage, identification, retrieval of knowledge and more importantly, to provide a more SQL-like query language (see examples of KEE and ART queries given above).

4.73 THE RELATIONAL DATA MODEL V THE SM's FRAME-BASED MODEL

The relational data model has some similarities to that of the frame-based representation developed for the SM. A relation (or a table) in a relational database has tuples (rows), and attributes (columns) that have to be of the same class (or type in procedural perspective). Relations may share certain characteristics or properties

through attributes of a common domain. These attributes can be used to create more relations (Date 1986, Beynon-Davise 1991).

Like relations in a relational database, the SM's knowledge base comprises hierarchies which define different classes of concepts. Concepts in a hierarchy share the general properties defined by generic objects (or frames) in that hierarchy.

In contrast, though, to relations in a relational database, frames are much more flexible and facilitate richer representation. Large numbers of subsets (sub-trees) of frames are organised in such a way as if relation instances were structured into a multi-level tree. A normally small number of offspring at each node facilitates browsing, and abstracting upwards of common attributes (columns), and their inheritance. In frame-based knowledge base systems procedural attachments are easily implemented. Furthermore, a frame-based representation allows exception, whereas in a relational database it is fundamental for every entry of a column (attributes) to be of the same kind.

The manipulative part of the relational model consists of a set of operations known collectively as relational algebra. The result of any retrieval operation in relational algebra is always another relation. Each operation takes either one or two relations as its operands and produces a new relation as a result. The three fundamental operators of the relational algebra are selection, projection and join.

em-no	name	position	salary
01	Jon	programmer	20k
03	Ian	proj-mang	14k
02	Masoud	programmer	21k
04	Terry	proj-mang	24k
06	Dari	analyst	10k
05	Neda	programmer	19k

Figure 4.10, a simple relation 'employee'.

The selection operation creates a subset of all the tuples in a relation that satisfy certain conditions. A selection on the relation shown in figure 4.10 can ask for the name of all of those employees whose salary is greater than £20,000. A SQL-like query of this example is as follows :

SELECT FROM employee WHERE salary ≥ £20,000 -> high-earners

The result is then returned as a relation called 'high-earners'.

The relation given in figure 4.10 can be presented in the SM as a hierarchy, called 'Employee', where each employee is represented by a frame containing the name, position, salary and employee's number as shown in figure 4.11. A query similar to that of selection in a relational model can be made to ask for the name of those employees whose salary ≥ £20,000.

The query format may be as follows :

```
((employee) ;the hierarchy's name is given by the user
   (?frame-names
     (salary (value (≥ £20,000))) )  )
```

The result of this query, (Terry and Neda), is returned to the user.

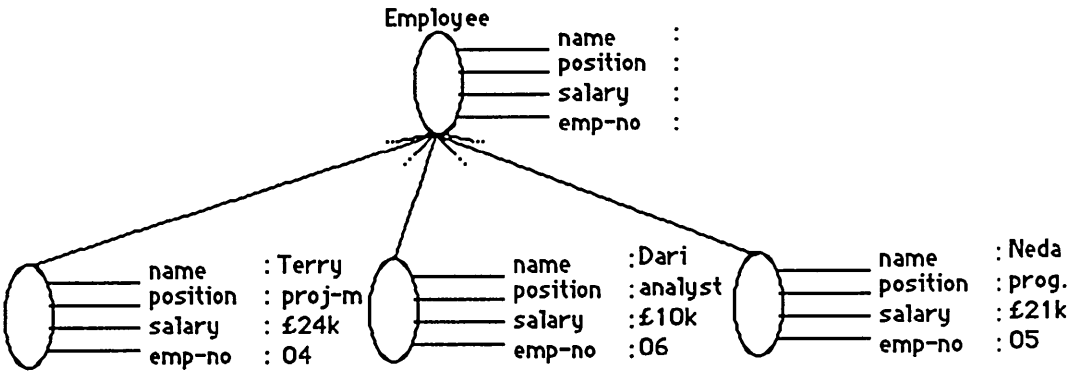


Figure 4.11, the representation of a simple relation in frame-based hierarchy.

The operations involving projection create a subset of all the attributes of a relation. In the example given in figure 4.10, we can ask, for example, for the name and the position attributes. A SQL-like query of this example may be :
PROJECT name, position, FROM employee -> positions

In return we will have a new relation called 'positions', with the attribute 'name' and attribute 'position'. The tuples in this relation would consist of all the employees that were given in the relation shown in figure 4.10. To implement projection in the frame-based hierarchy given in figure 4.11, the same query made for selection can be employed but this time we can ask for the name and position of all the employees represented in the 'employee' hierarchy. The query's format may be as follows :

```
((employee) ;the hierarchy's name is given by the user
   ( ?frame-names
     (position (value ?v2)))  )
```

The result of this query is returned to the user, which is as follows :

```
( (Terry (position (value : project-manager)))
  (Dari (position (value : analyst)))
  (Neda (position (value : programmer))) )
```

The third major operation in relational model is join. A complete class of join operators is called theta-join, where theta[7] stands for comparators such as equality, inequality and others (see Codd 1991, pp74). The theta join generally is referred to as join, and employs two relations as its operand. The condition expressed in the join operator involves comparing each value from a column of the first operand with each value from a column of the second operand.

The most utilised forms of join are equijoin (the theta join operator based on equality) and natural join. In these types of join, as mentioned above, two relations are taken as the operands and a new relation is produced (Deen 1985, Codd 1990, McFadden 1991). The new relation consists of all the rows obtained by concatenating each row of the first relation with all the rows of the second relation that have the matching value under the common domain. The columns in the new relation includes all the columns from both relations. The difference between equijoin and natural join is that the common column in the natural join is presented only once, where in equijoin the common columns appear redundantly in the resultant table. Furthermore, in both natural join and equijoin, a row in a relation is excluded from the resultant relation if it does not have a matching value in the common column of the other relation[8].

The following is a simple example given to demonstrate the operations involved with both natural join and equijoin. A query can be made as follows:

find all the employees and warehouses that are located in the same city.
Or a SQL-like query : Join A and B Where City-1=City-2 -> result

Empno	Ename	City-1
E107	Terry	London
E912	Dary	Leeds
E239	Ian	Leeds

Figure 4.12, relation A.

---

7 Theta comparators can also be applied to the select operator.

8 In another type of join; outer join, rows that do not have matching values in common columns are included in the resultant table.

Whouse	City-2
W1	Both
W34	Leeds
W92	Leeds

Figure 4.13, relation B.

Empno	Name	City-1	Whouse	City-2
E912	Dary	Leeds	W34	Leeds
E912	Dary	Leeds	W92	Leeds
E239	Ian	Leeds	W34	Leeds
E239	Ian	Leeds	W34	Leeds

Figure 4.14, the result of equijoin on relations A and B.

Empno	Name	City-1 =City-2	Whouse
E912	Dary	Leeds	W34
E912	Dary	Leeds	W92
E239	Ian	Leeds	W34
E239	Ian	Leeds	W34

Figure 4.15, the result of natural join on relations A and B.

The theta join of two relations can be conceived as a subset of the Cartesian product of those relations. In the Cartesian product of two relations, there is no given common column, and each and every tuple of the first relation is concatenated with each and every tuple of the second relation. In other words the resultant relation is the set of all tuples from both the operand relations of the degree m+n, where m is the number of columns of the first relation and n is the number of columns of the second relation (in the example above, the Cartesian product of A and B would be a relation of degree 5 with 9 tuples). It is possible, however, the equijoin given above can be replaced by Cartesian product of relations A and B, followed by the selection of those rows for which the condition City-1 = City-2 is met.

In serial machines, the operations involved with the Cartesian product consume a substantial part of the memory, disk space and channel time. This is also true, to a lesser degree, with the operations involved with join, and in many applications join is

replaced by a combination of less computational intensive operations such as select and project. As a result, in serial machines Cartesian product is not implemented as an explicit separate operator.

In the SM, as mentioned above, each relation can be presented as a hierarchy of frames in a frame-based knowledge base, where each frame in that hierarchy represents a tuple of that relation. In figures 4.16 and 4.17, relations 'A' and 'B' from figures 4.12 and 4.13, are shown respectively as a frame-based hierarchy. Note that the slot-name 'City-name' is the same for each frame in every hierarchy that inherits it, whereas in figures 4.12 and 4.13 the attribute names used are 'City-1' and 'City-2'. Although these attribute names are different, they are from the same domain and may have the same values.

Note also that the examples given in this section are simplified, mainly to provide a clear demonstration of the similarity between hierarchies and relations. A typical hierarchy, similar to those used in the SM's knowledge base, consists of several levels and it is highly likely that each frame will inherit a larger number of slot-names as we move down those levels. However, amongst all the levels in a hierarchy, it is most likely that only the leaf-node frames will have complete pairs of slot-names and their values. Thus, to compare a hierarchy with a relation, we may assume that each tuple of a relation is similar to a frame in a given hierarchy. This implies that the frames at the higher level, which have a lesser number of slot-names, can be regarded as tuples with a reduced set of domain-names.

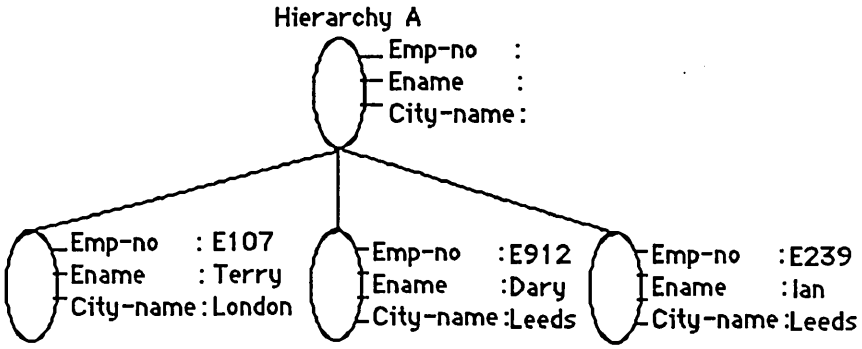


Figure 4.16, a simple relation 'A' is represented as a hierarchy of frames.

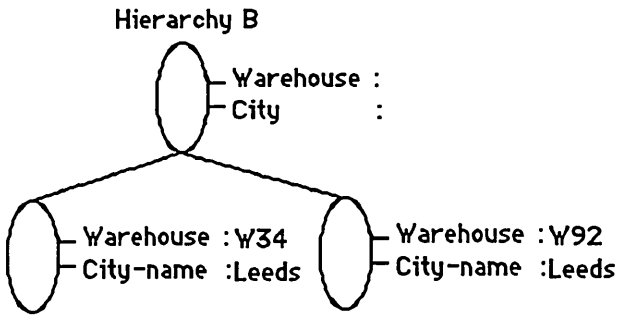


Figure 4.17, a simple relation 'B' is represented as a hierarchy of frames.

In the relational model, the columns to be compared are indicated explicitly in the join command (Codd 1990, and see examples above). Therefore, to make join-like queries in the SM, we would have to explicitly name the common slot-names shared amongst frames in different hierarchies.

As an example, a query can be made as follows:
find all the employees and warehouses that are located in the same city.
This query can then be represented in the following structure:

```
( (hierarchies 'A' and 'B')
  (?frame-names
  (City-name (value ?V1)) ) ) ;the common slot-name
```

In return we would have the following frames:

```
((Frame-name : Terry      ((Frame-name : Dary ((Frame-name : Ian
  (Emp-no : E107)           (Emp-no : E912)      (Emp-no : E239)
  (City-name : London))     (City-name : Leeds))  (City-name : Leeds))
((Frame-name : W34        ((Frame-name : W92
  (City-name : Leeds))      (City-name : Leeds))
```

The first frame in this group; Terry, does not share the same value as the others.

Those frames that share the same values for the City-name may be viewed as a set of tuples of different relations that will have to be concatenated in order to produce their products. The result may then be as follows:

Frame-name	Emp-no	City-name	Warehouse	City-name
Dary	E912	Leeds	W34	Leeds
Dary	E912	Leeds	W92	Leeds
Ian	E239	Leeds	W34	Leeds
Ian	E239	Leeds	W92	Leeds

The actual concatenation may be done after all the appropriate frames are identified and returned to the controller. Thus, to provide join, an extra layer of operations should be added to the SM's computational model. In appendix E, several options are suggested. The operations involved in each option, the advantages, disadvantages and the suitability of each option to the architecture, and the existing computational model of the SM, are also examined. Ultimately, a conclusion is reached that the computational model of the SM is most suitable to operations involved with select, and that the time penalties for join will increase but not as much as in serial machines.

5.1 INTRODUCTION

In this chapter, the simulation of the SM, its components (figure 5.1), their operations, and the components and data structures developed for the simulation program, are discussed.

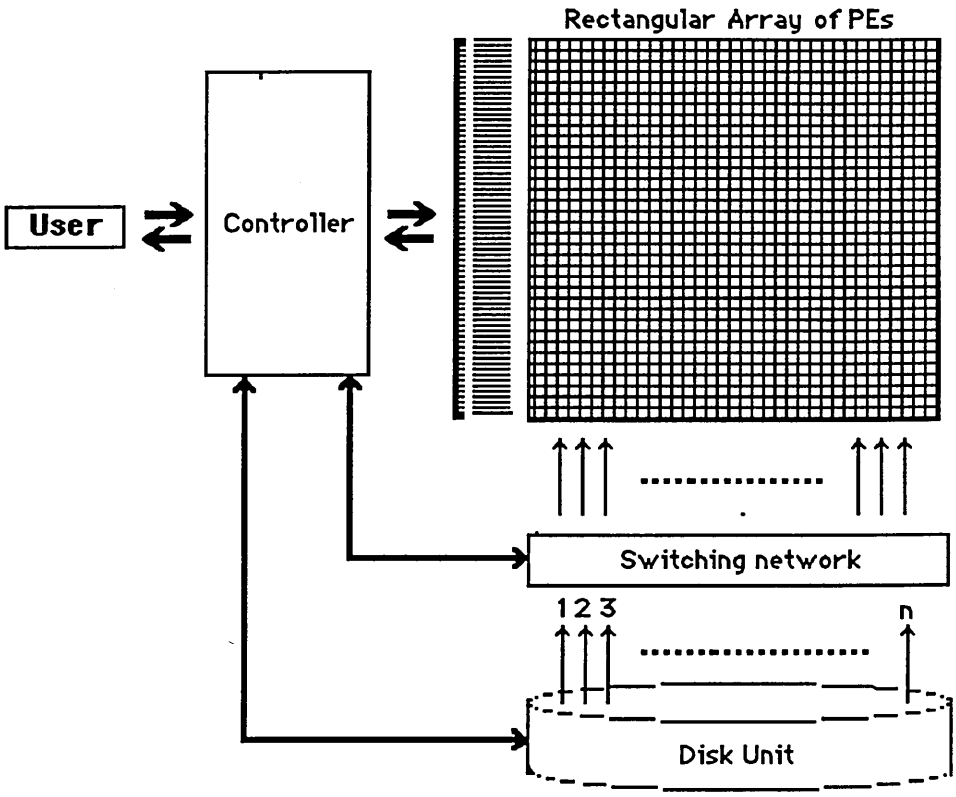


Figure 5.1, the block diagram of the SM system.

A simulation has been developed to investigate the behaviour of the SM system. This investigation is mainly focused on the parallel execution of the SM's frame-based knowledge representation language to determine possible advantages and disadvantages of this type of parallelism (see chapter 6). The central objective of the simulation is to find out whether the parallel execution of the language would result in faster processing.

The computational model of the SM's language consists of two main components; the knowledge base, and the distributed inference engine. The knowledge base resides on a parallel disk subsystem and consists of a large number of interconnected frames, where each frame represents an object. In the course of parallel propagation of

messages, in each PE, the inference engine performs inheritance of properties and pattern matching operations.

The SM's model of computation is based on downward transitive closure in which the messages are propagated from higher level frames to lower level frames. In this method, the propagation starts by the controller injecting a packet at each root-node in every mapped hierarchy. Initially, the contents of each packet is the same, but as it is passed from one PE to another and as a result of inheritance operations, its contents are modified.

A query language has been developed to enable the user to interrogate the SM's knowledge base. The type of queries available to the user can broadly be divided into two categories : frame-related conjunctive queries and domain-related conjunctive queries.

The simulation is written in ExperLisp which is a Lisp dialect that has been developed for Macintosh machines (Ritz 1987). The code for the simulation is presented in appendix A. In this chapter, various components of the simulation including queries and their overall operations are discussed

5.2 OVERALL OPERATIONS OF THE SIMULATED SM

The overall operations of the simulated SM are as follows :

1 - The controller creates a query-frame (if necessary) while interacting with the user. An example of a particular type of query-frame is as follows :

```
(object-name ((slot1 (value ?v1))
              (slot2 (value ?v2))
              (?slot (value a-value)) )
```

Each component of the query-frame can be either an actual value or an unbound variable, indicated by a heading question mark.

2 - The hashing algorithm takes the object-name and returns its equivalent hashing value.

Note : there is a type of query available in the SM in which the user can request the name of an object by providing full or partial information of that object. In this case, the hashing algorithm is not used. But instead, the user is asked to provide the name of

the relevant hierarchy containing the appropriate frame. In this case step number 3 is ignored.

3 - The hashing value of the object-name is matched against the hash-table (disk_index) which contains the hashing values of all the objects that exist in the knowledge base. This action not only determines whether the object exists in the knowledge base, but also finds its position on disk.

4 - By now the controller is able to retrieve the whole hierarchy and map it onto the rectangular array of PEs. While the mapping operation is in process, the controller creates a table which is used in conjunction with the disk_index. In this table (controller_copy), information is given concerning the logical and physical relations between the mapped frames, and the PEs that they reside on. The controller uses the controller's copy to find out the position of a frame in the array of PEs, and also its relation to other frames in the hierarchy.

5 - The controller initiates the propagation by injecting a packet at each injection point (root-node). There may be several injection points, in multiple hierarchies and domains, which may result in further retrieval and mapping operations.

Note that the distance between an object and any of its ancestors is called the "semantic distance". These distances are traversed while performing inheritance and matching operations. Since the propagation starts from the injection points, the various paths taken in the propagation are semantic distances from root-nodes to different objects.

6 - As the propagation of data packets progresses, parallel inheritance of properties and pattern matching operations are performed by all the PEs in the array until either appropriate conditions are satisfied or the propagation arrives at the leaf-nodes.

5.3 COMPONENTS OF THE SIMULATION PROGRAM

There are five major components that constitute the simulation program; initialisation, interface (provided by Menu), query-system, ADT (Application Development Tool- provided by CreateFrame), and graphics/output (see figure 5.2).

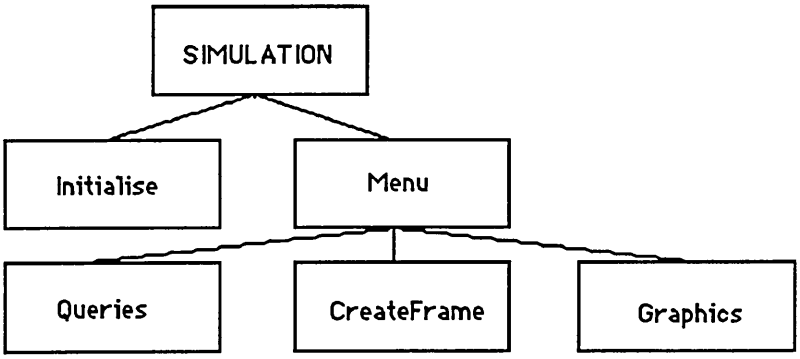


Figure 5.2, The main components of the simulation program.

These components have been linked together by an overall algorithm of the simulation which is given below :

```
INITIALISE
  Set-up MENU
  Set-up global variables
  Set-up arrays and other data structures
  Compile all the functions

IF   Queries    THEN
        IF Query_Object     THEN
              IF     Query_Type_One      THEN  call Query_Object_1
              IF     Query_Type_Two      THEN  call Query_Object_2
              IF     Query_Type_Three    THEN  call Query_Object_3
        ELSE         Return to MENU
        ENDIF
        IF Query_Domains  THEN
              IF     Query_Type_One      THEN  call Query_Domain_1
              IF     Query_Type_Two      THEN  call Query_Domain_2
        ELSE  Return to MENU
        ENDIF
ELSEIF  CreateFrame          THEN
        IF     Creating_Frame            THEN  call Create_Frame
        IF     Add_Slots                 THEN  call Add_Slots
        IF     Delete_Slots              THEN  call Delete_Slots
        ELSE  Return to MENU
        ENDIF
ELSEIF         Graphics        THEN        call Graphics_Package
ENDIF
```

In each interrogation, the row output produced by the simulation is a set of 2 output files. The first file contains the time-penalties for knowledge manipulation and communication (ie message passing; inheritance/matching operations) performed in a particular run[1]. The second file[2] contains various prompts produced at run time stating the stages involved. The time-penalties produced in the first file are utilised by the simulation's graphics package to produce a series of graphs. The second file contains information related to the propagation and its paths, packet injection, inheritance and relevant mapping operations. The graphics package together with all the output files and their evaluation/analysis is discussed in chapter 6.

5.4 INITIALISATION

The simulation starts with the initialisation process which invokes the simulation's interface. The interface, a pull-down menu, facilitates access to the query system, ADT and graphics.

The initialisation stage begins by loading files which contain all the lisp functions developed for the simulation program. This process invokes the menu system, which as mentioned above, acts as an interface by enabling the user to access the simulation for querying the system, developing an application, or for drawing graphs illustrating each interrogation.

After all the simulation's program codes have been compiled, a series of global variables and arrays are initialised. These variables contain information that are used for various operations involved in the simulation and the arrays are used to represent some of the components of the simulated machine. The main global variables and data structures used in the simulation are as follows :

List_of_clashes : an association list which contains the list of clashes at the time of hashing process (if any).

Coords : a global variable that contains the contents of the file "coordinates" representing coupled coordinates of a rectangular array of PEs and the Graphics-screen.

The_Propagation_Paths : this is a global variable which contains the paths of propagation at each run.

Free_maphistory : represents the largest level-number of the most recent mapped hierarchy. When it is bound to nil it means that there has been no previous mapping operation.

---

[1] Each file, accompanied with its corresponding graphs, is presented in appendix D.

[2] These files are presented in appendix C.

disk_array : a 2D array representing the disk (40 * 5).

controller_copy : a 2D array representing the controller's copy (2000 * 4).

disk_index : a 2D array representing the disk_index or hashing table (2000 * 4).

PEs : a 2D array representing the rectangular array of PEs (5 * 40).

5.5 INTERFACE

The menu system is provided as an interface between the user and the simulation. Figure 5.3, shows a screen dump of a simulation run where the menu can be seen.

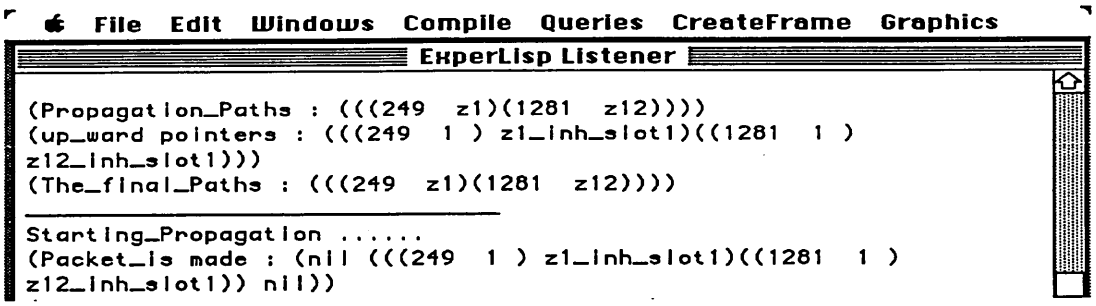


Figure 5.3, a screen dump of a simulation run.

In the menu the following options are given :

Queries

- help
- query_objects
- query_domains

CreateFrame

- creating_frame
- add_slots
- delete_slots

Graphics

- do_graphics

The "Queries" option enables the user to query the system about objects and their relationships in the knowledge base. The "CreateFrame" sub-menu, is a part of the ADT which assists the user to develop a new or modifying an already existing knowledge base.

5.6 QUERYING THE SYSTEM

In chapter 4, all the queries available in the SM were described in detail. Here, the operations and the software developed for each query are discussed. But before we start

these discussions, let us have a brief resume of the processes involved in the query-system.

The operation starts with the user choosing an appropriate query type for interrogating the knowledge base. In most cases, all queries require a query-frame, which is created by the controller while interacting with the user. The query-frame may or may not contain the name of the frame being queried, with all or some of its slots and possible slot-values.

The propagation starts by the controller injecting a packet at each injection point (root-node). The structure of each packet is as follows :
([query-slots] [upward-pointers] [inherited-slots]).

Packets at the root-nodes of all the mapped hierarchies start moving down from one frame to the next one in the path, until appropriate conditions are met. Each PE that contains a frame in each path will check the contents of the packet that it receives. The checking involves all the contents of the packet : query-slots, upward pointers and the inherited slots. If, in the list of upward pointers, there is a hash number which is the same as the frame's hash number that is being processed, then all the necessary inheritance operations will be performed on that frame. This approach adds further efficiency by reducing the time spent on unnecessary pattern matching and inheritance operations. The result of inheritance operations will be added to the packet, and a copy of that packet will be added to the PE before it is passed down the path to another PE.

In figure 5.4, a diagram showing the components of the query-system and their relationships, are presented.

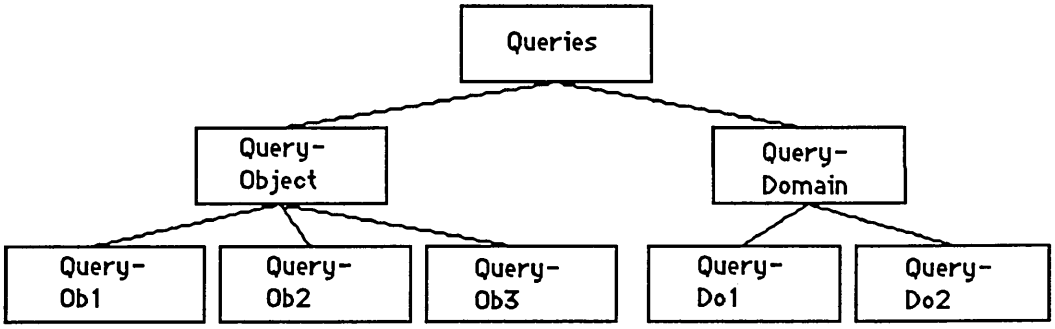


Figure 5.4, the query system.

Notice that as discussed in chapter 4, the underlying computational model in the SM is based on "cyclic downward transitive closure of the messages", which has been utilised as the building block for every query available in the system. The key element that has played an important role in implementing such an approach is the unified data

structure used in the SM; frames. Frames represent the knowledge in the knowledge base, its related information, procedures, and queries. And because of it's distinct components - slots and their values- and the ability to represent both knowledge and query in the same format, it is possible to have a basic computational model upon which different query-dependent operations are superimposed. In this model, the components of a frame; frame's name, slot-names, slot-values, can all be matched with any other frame's. Frames, for example, from the knowledge base can be matched against a particular query-frame. Whether the aim is to find the frame's name, or to find approprite slot-names, slot values or both, or/and its relationships with other frames, a query-frame can be made to provide such requirements. After the appropriate inheritance and pattern matching operations are performed, the unbound variables representing the missing parts (or required data) will be instantiated to real values and returned to the user.

Thus, despite whatever the query-type, in the SM there is an underlying computational model, which is used as the basis for all operations.

It is important to note that the graphs shown in the following sections (run-time graphs) represent only the propagation paths in which the frames, that are related to the given query, reside. Otherwise, as mentioned above and in chapter 4, the parallel execution of the SM's language involves all the PEs in the array that contain a frame. In other words, the paths taken for propagation in each run would involve every available link between frames that have been mapped to the array of PEs in the SM. Therefore, the highlighted paths seen in figures 5.5, 5.6, 5.7 and 5.8 are only shown because they contain all the related frames to the given query, not because they are the only propagation paths at each execution. A discussion on graphs produced at run-time is presented in chapter 6.

5.61 FRAME-RELATED CONJUNCTIVE QUERIES

In this type of query, the user has three options :

a) Asking for a full definition of a named object.
b) Asking to complete partially given information on a named object.
c) Asking for a frame-name by giving its partial or full definition.

In the first option, the user is required to give the object name only and a complete definition of that object will be returned. In the second option, the system will interactively create a query-frame, of which some slots or slot-values are missing. The system will then return the query-frame with those missing parts filled. In the

third option, the user provides some partial information or a full definition of a frame, and in return, the system will provide the user with the name of that object. Some of the test runs for this type of queries are shown in appendix D.

5.611 OPTION ONE : QUERYING FOR A FULL DEFINITION

The user can ask for the full definition of an object by choosing the first option in query-object. Here, only the object name is given and the system will return a fully defined frame which has undergone all the necessary operations including inheritance of properties.

When the object's name is given, by calling the hashing functions, the disk-position of the frame will be determined (if the given object does exist in the KB). After the position of the given object is found, it will be retrieved from disk.

A series of functions at this stage determine whether there is a need for mapping of the portion of the knowledge base that contains the query-object (ie it has already been mapped). These functions are also called at various stages of the interrogation, for example, in the case of multiparentage where there are more than one parent for a particular frame, some parents can be from other hierarchies that have not been mapped to the system.

The system will then determine all the injection points from which the propagation is going to start. These injection points, are also used to determine the propagation paths that will be used in the simulation for performing inheritance and matching operations.

The operations involved in this type of query :

-Get the object-name (from the user)
-Determine all the injection points and propagation paths
-Check if each injection point has been mapped
-Get all the upward pointers
-Start propagation from the injection point of each path
-Continue until the end of all the paths have reached.

For each path
Create packets[3]

---

3 Note that the contents of a packet are as follows :

Inject packets at the injection point of that path

set up all the timedelay parameters

Check mapping for each frame in the path

For each frame

The inheritance operation is performed on each frame as follows :

IF a-frame has parents and DOES require inheritance,

DO inheritance and COPY a packet and PASS it to the other PE.

IF a-frame has parents and DOES NOT require inheritance

COPY a packet and PASS it to the other PE.

IF a-frame has ONE parent and requires inheritance

DO inheritance and COPY a packet and PASS it to the other PE.

OTHERWISE a-frame has NO parents and DOES NOT require inheritance

COPY a packet and PASS it to the other PE.

Matching operation in each PE

If the inheritance is required (See above), there will be a series of matching operations between all the inheritable slots of the frame in the path, and all the inherited slots (the third part of the packet) from higher levels of the hierarchy. As a result of these matching operations, all the information (slots and their values) situated in the third part of the packet, will be updated.

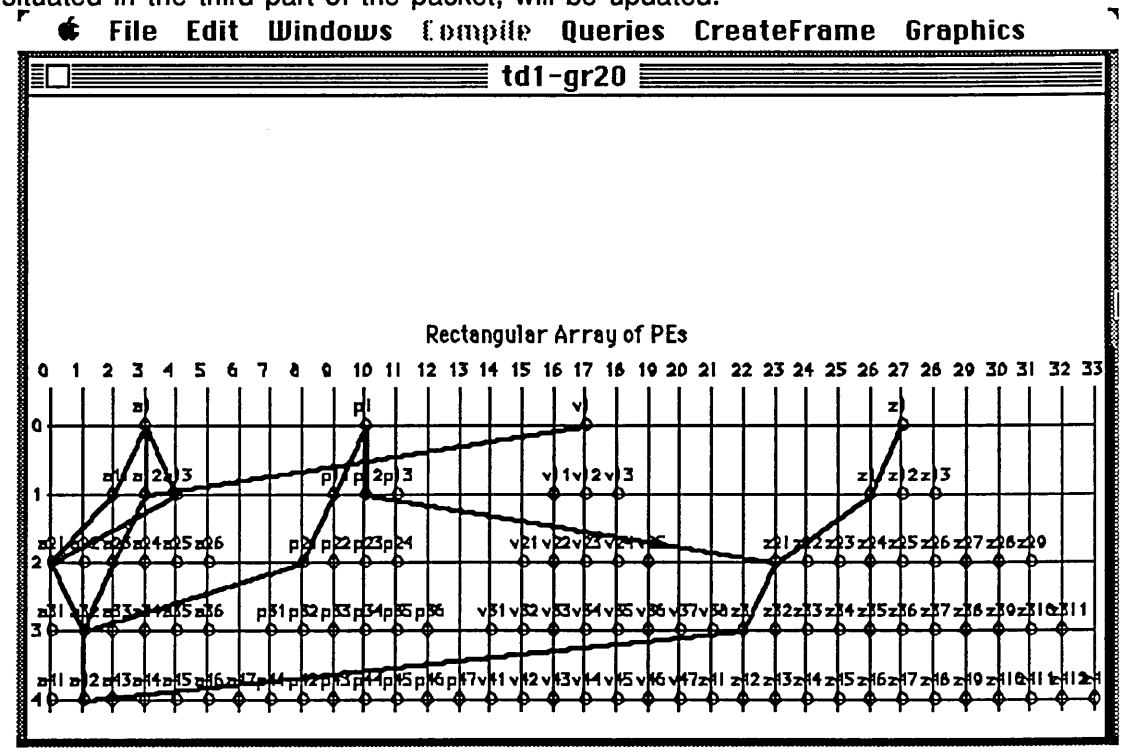


Figure 5.5, a screen dump of a run-time graph showing all the relevant propagation paths.

((query slots) (upward pointers) (inherited slots)) but in this type of query, the query-slots is nil.

In figure 5.5, a screen dump of the run-time graph (graph number one in appendix D) for the given query in the above example, is shown. As mentioned above, this graph only shows the relevant propagation paths converging towards the query-frame; 'a42'. Note that on all the relevant frames in these paths, inheritance operations will be performed. The injection points in figure 5.5 are; a1, p1,v1 and z1, from which the propagation starts. In this figure, only the paths relevant to the given query are shown.

5.612 OPTION TWO : QUERYING WITH PARTIAL DEFINITION

In this option, the user can ask for a subset of slots/values of a frame. As in the first option, when the object's name is given, by calling the hashing function, the disk-position of the frame will be determined. After this operation, a query-frame is made, by which the user enters all the slots (with possible values) and the mapping functions will determine if it is necessary to perform mapping or not.

Apart from making a query frame and performing inheritance operations only for those slots that have been specified in the query-frame, all the operations are similar to that of option one. The operations involved in this type of query are as follows :

-Get the object-name (from the user)
-Create the query-frame
-Determine all the appropriate injection points and propagation paths
-Check if each injection point has been mapped
-Get all the upward pointers
-Optimise the paths
-Start propagation from the injection point of each path
-Continue until the end of each propagation path.

For each path
  Create packets[4]
  Insert a packet at the injection point of that path
  set up all the timedelay parameters
  Check mapping for each frame in the path

---

[4] Here in this type of query, the packet structure is as follows :
((query slots) (upward pointers) (inherited slots)). Note that unlike the first option, there is a set of query slots.

For each frame

The inheritance operation is performed on each frame in the path as follows :

IF a-frame has parents and DOES require inheritance,

DO inheritance and COPY a packet and PASS it to the other PE.

IF a-frame has parents and DOES NOT require inheritance

COPY a packet and PASS it to the other PE.

IF a-frame has ONE parent and requires inheritance

DO inheritance and COPY a packet and PASS it to the other PE.

OTHERWISE a-frame has NO parents and DOES NOT require inheritance COPY a packet and PASS it to the other PE.

Matching operation in each PE

In this option, there are facilities which enable the system to avoid unnecessary inheritance and matching operations. When the propagation paths have been determined, according to the upwards pointers, only those paths will be chosen. If inheritance is required (See above), there will be a series of matching operations between all the inheritable slots of the frame in the path and all the inherited slots (the third part of the packet) from higher levels of the hierarchy. As a result of these matching operations, all the information (slots and their values) situated in the third part of the packet, will be updated.

As in figure 5.5, figure 5.6 shows the injection points from which the relevant propagation paths have started and converge towards the query-object.

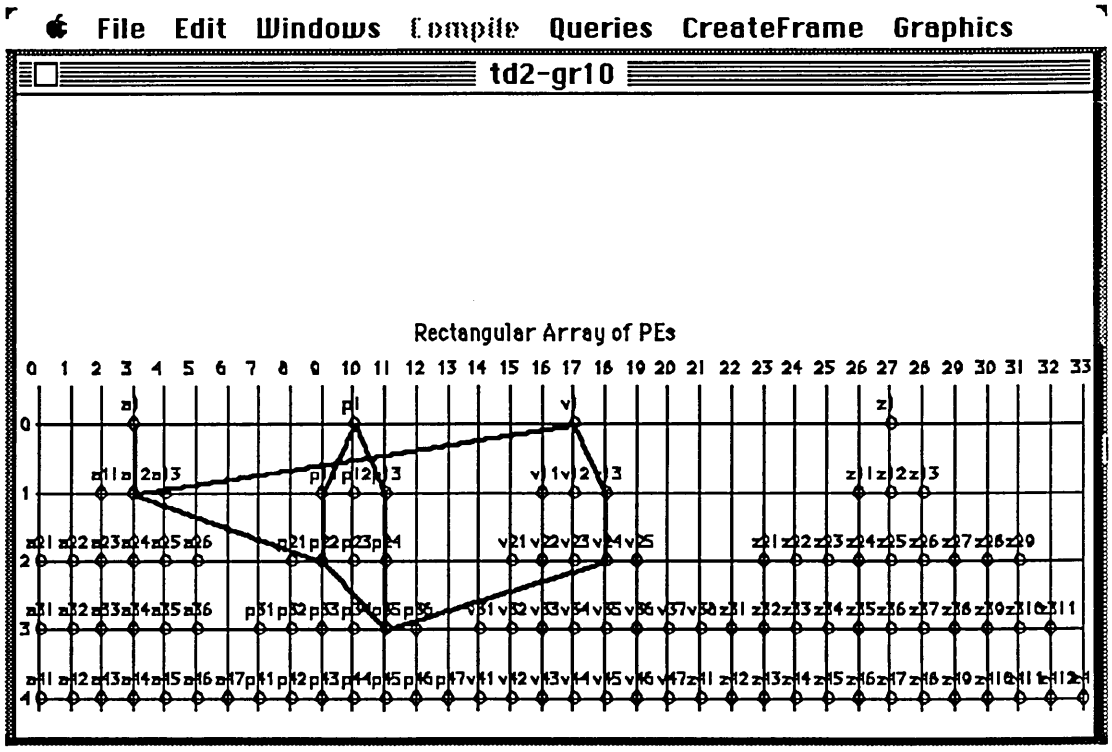


Figure 5.6, a screen dump of a run-time graph showing all the relevant propagation paths.

5.613 OPTION THREE : ASKING FOR A FRAME'S NAME

In this option, as discussed in chapter 4, the user can ask for the name of a frame by providing its full definition, or a subset of its slots/values. It is also possible for the user to provide only a partially defined subset of slots and slot-values. That is, some of the slot-names are given without values, or some of the values are given without their slot-names. In such cases, the system provides unbound variables, which in the course of propagation will be bound to appropriate values.

Unlike the previous options, when the query is made, the controller cannot utilise the hashing mechanism to find the position of the object in the knowledge base and on disk. The controller, instead, identifies the appropriate hierarchies by asking the user for their names. An example of such a query may be as follows :

```
((v-hir)   ;this is the name of hierarchy given by the user
     ((a1_inh_slot4  (value  ?V1))
      (a12_inh_slot1  (value  ?V2))
      (v23_inh_slot1  (value  ?V3)) ))
```

After the appropriate hierarchies are mapped to the system, the remaining operations for this type of query are the same as options one and two. The main difference is that the query-frame has no frame-name but contains all the given slots with their values. As it was mentioned at the beginning of this chapter, this is due to the unique data structure of frames and the SM's underlying computational model which is the basis for the operations involving any available query in the system.

The slots, slot-values and, in their absence, variables representing them, will all be part of the query-frame. In the course of propagation, they are matched against the properties of frames in the mapped hierarchies. In these hierarchies, if there is a frame which contains the same information as that of the query-frame, its name will be returned to the controller.

As in option two, in this type of query the controller starts making a query frame and performing inheritance operations only for those slots that have been specified in the query-frame; all the operations are similar to those of option one. The operations involved in this type of query are as follows :

-Get the names of relevant hierarchies (from the user)
-Create the query-frame
-Determine all the appropriate injection points and propagation paths
-Check if each injection point has been mapped

-Start propagation from the injection point of each path

-Continue until the end of each propagation path.

For each path

Create packets[5]

Insert a packet at the injection point of that path

set up all the timedelay parameters

Check mapping for each frame in the path

For each frame

The inheritance operation is performed on each frame in the path as follows :

IF a-frame has parents and DOES require inheritance,

DO inheritance and COPY a packet and PASS it to the other PE.

IF a-frame has parents and DOES NOT require inheritance

COPY a packet and PASS it to the other PE.

IF a-frame has ONE parent and requires inheritance

DO inheritance and COPY a packet and PASS it to the other PE.

OTHERWISE a-frame has NO parents and DOES NOT require

inheritance COPY a packet and PASS it to the other PE.

Matching operation in each PE

In this option, the system performs all the necessary inheritance and matching operations similar to options 1 and 2. In the inheritance operation there will be a series of matching operations between all the inheritable slots of each frame in the propagation path and the inherited slots in the packet (the third part of the packet). As a result of succcessful matching operations, all the slots and slot-values of the related frames, and slots and slot-values situated in the third part of the packet, will be updated. While the pattern matching and inheritance operations are being performed, if the given query-slots and their values match the contents of any frame in the path of propagation, its name will be returned to the user.

In figure 5.7 the propagation paths of all the frames that are directly related to the slots and slot-values given in the query-frame are shown. These paths, as mentioned before, are merely shown to put emphasis on the amount of operations performed on the related frames that they contain, in contrast to those frames that are unrelated to the given query. The matching and inheritance operations on unrelated frames is minimal. That is, as soon as there is an unsuccessful match between the components of the query-frame and any frame in the path, the whole process will be halted and the packet is moved down hierarchy to the next frame in the path. Thus, although at run-

---

5 In this type of query, the packet structure is as follows :
((query slots) nil (inherited slots)).

time the pattern matching and inheritance operations are being performed on every existing frame in the array of PEs, these operations continue only on those frames that have some common elements with the query-frame (these frames and their relationships are highlighted in figure 5.7).

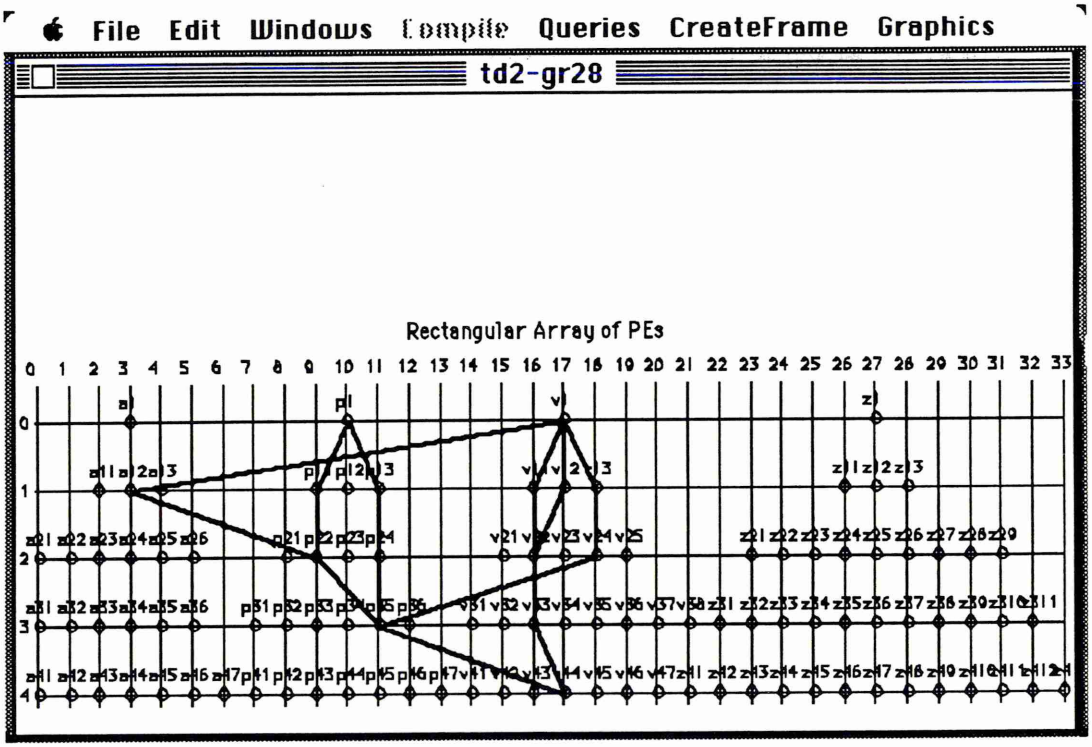


Figure 5.7, a screen dump of a run-time graph showing all the relevant propagation paths.

5.62 DOMAIN-RELATED CONJUNCTIVE QUERIES

There was a detailed discussion in chapter 4 of the two options available for domain related queries. Both in this chapter and in chapter 6, only option two is selected for discussion, testing and analysis. This is justified by the complexities contained in this type of query since the operations involved subsume the operations in option number one.

5.621 DOMAIN RELATED QUERIES OPTION TWO

This type of query enables the user to ask for a wide range of information about the relationships amongst frames in hierarchies, domains and the knowledge base. The user for example, can ask for all the frames that share one or more slots. In the absence of known frame-names, slot-names, or slot-values, variables may be given.

In this type of query, the controller may receive one or more object-names plus one or more slots with (or without) their values, and unbound variables representing

unknown shared slots or slot-values. The query-frame is created interactively by the user via the menu system and may have the following format :

```
((a23  a32  a34) ; frame-names
        ( (a32_inh_slot1  (value  ?v1))
          (?slot1  (value  (a32_inh_v8  a41_inh_v8)))
          (v1_inh_slot1(value(v1_inh_v1  a12_inh_v5  a23_inh_v6)))  ))
```

In this example the query-frame contains 3 frame-names sharing 3 slots. The first slot has a name and an unbound variable, which at run-time will be bound to a value. The second slot has no name and the third slot is complete. In the course of cyclic transitive clousure of data packets, after the inheritance operation is performed, the contents of this query-frame (as part of the packet) will be matched against each frame on the paths of propagation. In the case of a successful match, the variable ?v1 will be bound to appropriate value, which may have been inherited from higher level frames. If the match on the second slot is also successful, the unbound variable '?slot1' will be bound to its appropriate value. This query could take the following form:
Find "3 car manufacturing companies" selling cars with the following characteristics :

1- price is ?v1 pounds
2- ?slot1 4
3- engine-capasity is 2000cc

Another implementation of this type of query could have been made without giving the frame-names, whilst the user would have been asked to produce names for appropriate hierarchies. In chapter 4, several example of this type of query were given that show the range in which diverse queries can be made. However, the preliminary computations required for this type of query are as follows :

1- Check if any frame-names are given in the query, and if so,
2- create the query-frame.
3- identify, retrieve and map appropriate hierarchies to the array.

In the absence of frame-names in the query, the controller will ask the user for the name of hierarchies relevant to the query and will start from stage number 2.

As mentioned earlier, the underlying computational model in the SM is the same for every type of query available; a higher level computation is superimposed on it to meet the relevant requirements of the given query. For this type of query the following operations will be performed :

a) Start the propagation from the injection points. While the packets are moving down from higher level frames to their children, perform inheritance and matching operations

b) After the inheritance operation on each frame, by pairing up slot-names and slot-values in the frame with those given in the packet, check if all the given query-slots and correct values are contained in that frame. In the case of given variables, if the matching of either slot-name or slot-value has been successful, the variable will be bound to its appropriate value.

c) If all the query slots are matched successfully, the frame's name and relevant information will be passed to the controller.

d) This process will continue until the leaf-nodes are reached.

Note: While performing inheritance it is possible to discover that one of the offspring has multiparents. In such a case, the controller will perform further mapping and appropriate inheritance operations. Note also, the matching operations involved here are exactly the same as those used in other queries.

The summary of operations for this type of query is as follows :

-create the query-frame

-get the names of all the relevant hierarchies, if necessary

-provide appropriate information for storing time delay

-get all the injection points and propagation paths

-check mapping

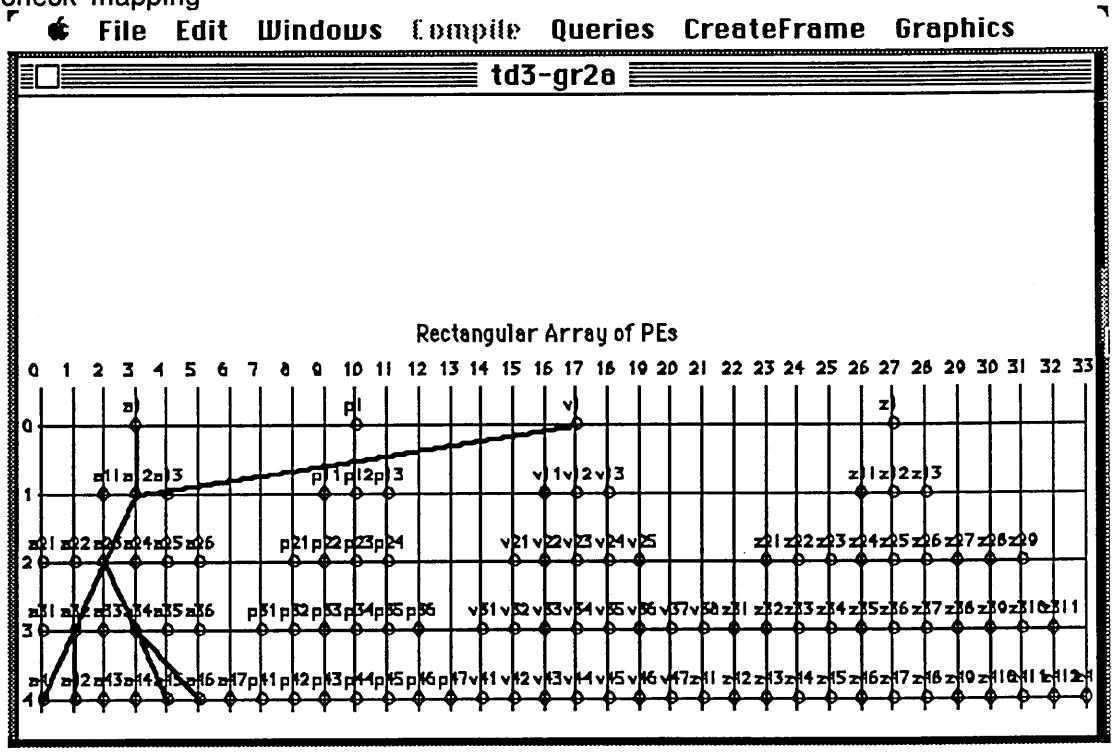


Figure 5.8, a screendump of a run-time graph for domain related queries type two.

Figure 5.8 shows all the relevant propagation paths in which frames relevant to the given query are situated.

For each path
-create a packet
-start propagation
-initialise all the parameters for the time-delay
-check mapping

Working on one frame
- IF a-frame has parents and DOES require inheritance
  do inheritance and add the packet to appropriate PE
- IF a-frame has parents and DOES NOT require inheritance
  add the packet to appropriate PE
- IF a-frame has ONE parent and requires inheritance
  do inheritance and add a packet to appropriate PE
- OTHERWISE a-frame has NO parents and DOES NOT require
  inheritance add THE EXISTING packet to appropriate PE

In each frame in the path, if necessary, inheritance is performed using the contents of the packet. It then matches the result with query-slots (with or without variables) and if all the query-slots are contained in that frame, the frame's name is passed to the controller. Otherwise, the frame will be ignored, and it will move down to the next frame on the path.

5.7 BENCHMARK KNOWLEDGE BASE AND ADT

In the simulation, an Application Development Tool has been developed to assist the user in the creation of knowledge bases, and in the modification of existing ones. This will provide speed in the development, better maintenance, and consistency within the knowledge base .

The knowledge base is stored on disks (see section 5.71) and it is assumed that at any interrogation, certain portions of it will be retrieved from disk, and mapped to the array of rectangular array of PEs. In chapter 4, a detailed discussion of the knowledge base was given. In this chapter, in addition to ADT, the benchmark knowledge base developed for the simulation will be discussed.

5.71 BENCHMARK KNOWLEDGE BASE

In chapter 4, it was said that the knowledge base in the SM consists of a large number of domains where each domain contains several hierarchies. In each hierarchy, frames represent objects of a particular domain. These frames are linked together according to

their relationships, thus forming a hierarchical structure in which lower level frames receive properties from higher level frames, by the performance of the inheritance operation. The relationships amongst frames are used as paths for propagation, which provides communication between frames. In this context, the communication is based on passing messages, which are contained in data packets, from higher level frames to lower level frames. Generally, these messages are used for the inheritance operation.

Therefore, the overall structure of the knowledge base and the structure of each individual frame have a direct and explicit relationship to the SM's distributed/replicated inference engine. To test the system, it is necessary to use a benchmark knowledge base which contains all the possible complexities of real-world knowledge. A benchmark was adopted and made suitable for the SM's system (Gray J. 1987), but it did not completely satisfy all the necessary requirements in testing the system and their analysis (eg lack of multiparentage, number of slots in each frame, etc.).

In order to encapsulate all these complexities in a small knowledge base, a benchmark , by using ADT, has been developed which is basically an abstract knowledge base and does not represent any particular application domain. This knowledge-base contains four domains (or hierarchies) which are heavily interlinked. There are two types of multiparentage; local and global multiparentage. The local multiparentage is concerned with those nodes that are sharing the same child in the same domain. In global multiparentage on, on the other hand, there are nodes from different domains that share the same child. The concept of multiparentage requires multi-inheritance operations and, therefore, imposes a certain amount of complexity and constraint on the propagation and inheritance operations. These constraints are necessary so that the system can be tested to its limit.

There are four hierarchies in the benchmark knowledge base having a1, p1, v1 and z1 as their root-nodes, respectively. Each frame, in the first hierarchy, contains or inherits substantial number of slot/values, eg "a1" (the root_node of the first hierarchy) has four slots that are inherited by all its offsprings. Frames at the lower levels of the first hierarchy will inherit a large number of slots from local/global parents. This is useful for studying the characteristics of transferring a large amount of data from one node to another, in the SM's rectangular array of PEs. However, the size of each frame gets smaller in the second, third, and fourth hierarchies respectively, but the multiparentage relationships are increased.

Each frame in the benchmark knowledge base has three groups of slots; id_slots used for identification, ind_slots representing the individual characteristics of a frame and inh_slots for inheritable slots. In each inheritable slot, there is an inheritance constraint facet which provides certain conditions, on the way properties are inherited. These constraints are; default, union, intersection and override.

In the id_slots of a frame that has more than one parent, a list is given which contains the name/pointer of every parent. The most dominant parent (or most related parent) is always positioned at the left hand side of the list thus, the first parent has the highest priority, and the last parent has the least priority (the parent on the right hand side of the list). This is important when inheriting properties from more than one parent. That is, in such situation, the slot/values of the parent with the highest priority will always have better chance to be inherited than those with a lower priority.

In the inheritable slots group, the positions of the slots comply to a certain ordering mechanism. The first slot in the list is inherited from the highest applicable level (ie the injection point), and the last slot is usually from the frame itself. Unlike individual slots, all the inheritable slots have pointers which are used to determine the source of inheritance. This ordering mechanism together with the pointers coupled with each slot, enable the overall controller to determine the injection points and propagation paths.

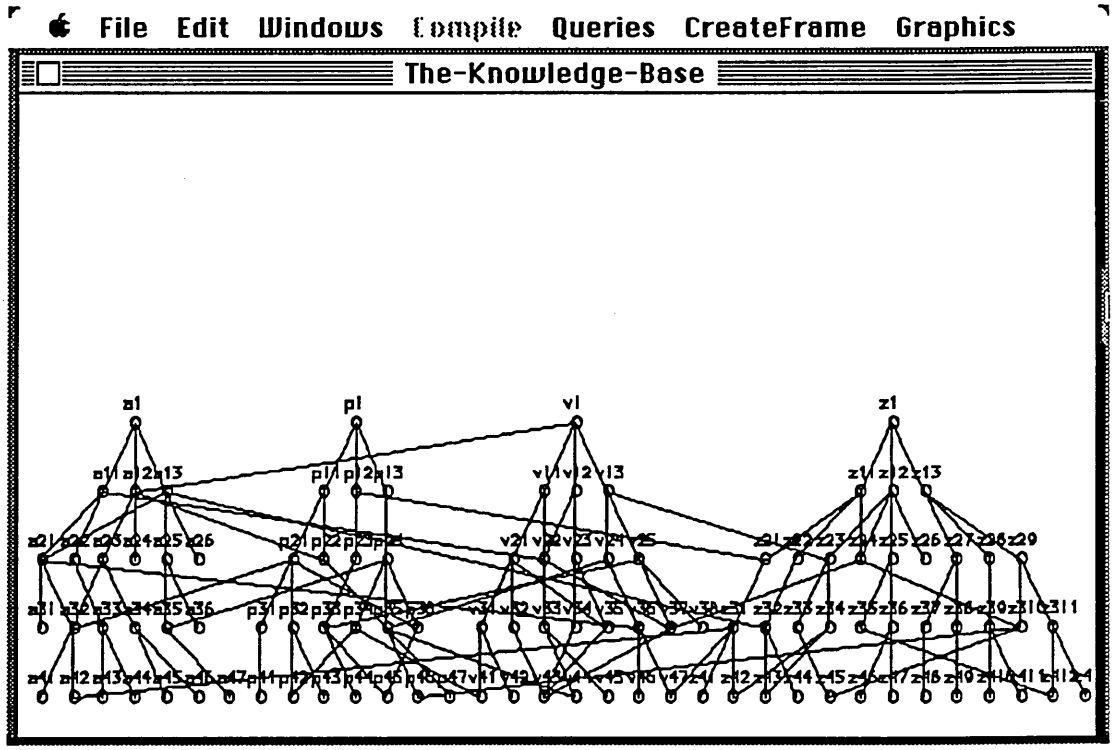


Figure 5.9, a screen dump of the bench mark knowledge base.

Note that, as was mentioned at the start of this chapter, there is another method to find all the injection points. That is, the controller can search the disk_index, disk and each consequent frame, to find the injection point. This method obviously takes longer than the ordering mechanism, but to ensure the ordering is always maintained may itself be a very difficult and time consuming task. In figure 5.9, a screen dump of a bench mark knowledge base that has been drawn by the simulation's graphics package is shown.

It was mentioned that the benchmark knowledge base was developed purely for the simulation. The knowledge base (see the complete version in appendix B), does not represent any particular domain. Instead it contains all the possible relationships and constraints that may exist in a real-world knowledge base, and it is geared directly towards the simulation, and all the tests associated with it. The amount of slots/values to be inherited, the size of data packets and the complexities imposed on inheritance operations by multiparentage and consequent combination of different packets, has provided a rich environment for testing the simulation.

5.72 APPLICATION DEVELOPMENT TOOL (ADT)

To develop a knowledge base, an application development tool will be required. This tool is a necessary part of any knowledge based system without such a tool, the task of creating each frame and performing the necessary inheritance operations, will be extremely tedious and time consuming. Certain parts of ADT, used for creating or modifying frames, are also utilised at run-time, while performing inheritance in every relevant PE.

The developer is an interactive toolkit which is used to create frames, and eventually, the whole knowledge base. By interacting with the user, it takes the name of the frame to be created and retrieves all the inheritable slot-names from its parent(s), unless the node being created is a root-node. If there are inherited slots, the user will have a choice to specify values and inheritance constraints for them. In addition to inherited slots, the user can add to the frame its own inheritable slots. The new frame will be stored on disk, at its appropriate hierarchical position.

In the case of any additions or deletions made to the knowledge base (ie slot addition or deletion), the control mechanism will traverse down the appropriate hierarchies and modify those frames that are affected. It is assumed that all the parallelism offered by the SM can by fully utilised in such operations. In ADT, the regulations and restrictions on inheritance operations are exactly the same as those in operation at run-time.

The ADT will help the user to create a frame and attach it to a certain part of the hierarchy. When the position is determined, ADT will identify all the possible parents of the newly made frame. The inheritance operation will be automatically performed on a default basis and a choice is given to the user to change the inheritance constraints to one of the other three possible options available in the system (override, union and intersection).

Note that, at the time of building the knowledge base, for every frame, only the slot names are inherited and their values will be inherited at run time. However, the system could have been developed to perform the complete inheritance either at the time of building the knowledge base, or at run time.

5.8 DISK-UNIT, HASHING, RETRIEVAL AND MAPPING

In the following subsections the disk-unit, the hashing algorithm and the hashing tables, the retrieval and mapping of appropriate hierarchies and the functions and contents of PEs are discussed.

5.81 DISK-UNIT

With the requirements of today's applications, it is essential for large systems to incorporate sophisticated disk subsystems, which offer a large amount of storage capacity with fast access and transfer time while providing fault tolerance and security. To meet these requirements, the disk-industry has taken a new approach by developing disk array subsystems with various characteristics denoted by RAID levels (Walder 1992). In RAID (Redundant Array of Inexpensive Disks) there are 5 levels: disk striping (level 0, generally not regarded as a level in RAID), disk mirroring (level 1), disk stripng with redundancy (level 2), parity bit checking (level 3), sector striping (level 4) and interspersing (level 5). In disk striping, data is written by system segment size, sequentially, from disk to disk in the array. That is, segment 1 is written to drive 0, segment 2 to drive 1 etc. Disk mirroring, in its simplest form, involves a single disk controller for two separate disks and while data is written to one disk it is automatically written to the mirrored disk. Disk striping with redundancy offers the advantages of disk striping, but with fault tolerance, and without the high level of data redundancy of mirroring. In the previous levels, error correction codes are used for data protection, whereas in level 3, parity bit checking is used, which has less overheads by using a single drive for error correction. In levels 2 and 3, a single data transfer block is split up to be interleaved across the data drives, whereas in level 4, the entire first transfer block is placed on the first drive, the second on the second drive, and so on. All the RAID levels discussed above, are

limited to single writes at a time (by using dedicated check disks), whereas in level 5, the error correction blocks are interspersed (spread) evenly across all disks along with the data. Because there is no dedicated check list, it is possible to perform multiple simultaneous writes/reads, which vastly improve input/output performance.

An appropriate disk subsystem for the SM should bear most of the characteristics offered by the various RAID levels, including a suitable method for data storage, disk access and data transfer. In addition to its more flexible utilisation of error correction block, level 5 may provide most of these requirements. In this level, similar to level 4, sector striping is used ie, a single data transfer block is placed on each disk in turn. This method of storage may be extremely suitable where all or some levels of a hierarchy can be placed on a disk at each interval. As processing power has increased due to the development of faster and more sophisticated processors, disk access time has been reduced to the point where manufacturers have reached a mechanical limitation on the reduction of access time while maintaining the system's reliability. A typical disk subsystem may have an access time of 10 - 18 mseconds. In RAID level 3 system, spindle sychronisation is incorporated resulting high data transfer rate, where searches on every drive are performed simultaneously. In most cases, where the access time plays an important role in the whole system, it is reduced to a minimum average (typically 10 mseconds) and if spindle synchronisation is used, it is regarded as a fixed time penalty at each disk access.

Thus, for the SM's large knowledge base, RAID levels 3 and 5 can provide a disk subsystem with a large capacity, fast access time and a high transfer rate. This requirement can be met by one of the commercially available parallel disk subsystems. In DIA (Disk In Array) for example, a number of magnetic disk drives are configured into an array, and run in parallel to achieve a very high transfer rate of 36 Mbyte/s. This sub-system has the storage capacity of 15 Gbytes, which can be increased to 120 Gbytes by adding extra disk drives to it (Oyama 1991). The disk unit in DIA consists of 8-inch magnetic disks operating as a single disk drive, a disk is used for the parity disk drive, and another disk is used for the spare disk drive. The unit incorporates spindle synchronisation providing simultaneous read/write facilities and reducing disk access time to 12 mseconds with a latency of 6.9 mseconds.

In the SM's simulation, it is assumed that the information is stored on a DIA-like parallel disk subsystem which may be read and then transferred to the rectangular array of PEs, via a switching network. The time penalties for disk access time and its relevant latency, and for the transfer rate for F6490 DIA, are incorporated in the simulation of the SM for its disk subsystem.

It should be noted that the operations involving storage, identification, retrieval and mapping of knowledge are complex, and at implementation level would require a substantial amount of time and research. In the simulation, a set of components have been developed to meet the initial requirements, and their time penalties have been considered.

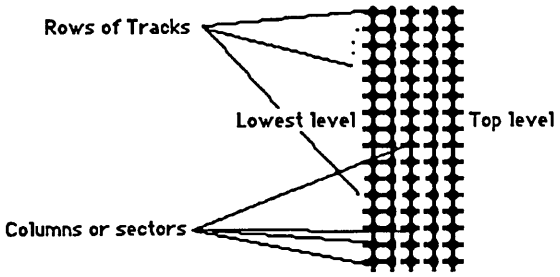


Figure 5.10, disk storage in the simulation.

In the simulation the data is stored in an array of two dimensions (disk_array), where rows and columns are regarded as tracks and sectors respectively. The storage organisation of hierarchies is shown in figure 5.10 where each frame is stored in a sector.

The lowest level of the sub-hierarchy is stored in the first column, ie the first frame will be stored in location (0 * 0) in the array, the second will be stored in (1 * 0) and so forth. Thus, the first column and n rows will, represent the lowest level of the first hierarchy. The second lowest level will be stored in the same manner but in the second column, and this process continues until all the levels of a given hierarchy are stored.

To store the second hierarchy we have to know the number of rows 'n' in the first column of "disk_array" used for the previous hierarchy, so that the lowest level of the second hierarchy can be stored in the first column starting from row n+1. When row n+1 is determined, we can start the same procedure to store the second and subsequent hierarchies.

Thus, when all the hierarchies are loaded in "disk_array", the first column will represent all the lowest levels of each hierarchy, and the second column will represent the second levels of all the loaded hierarchies, and so on. This representation is directly analogous to the tracks and sectors of a disk, as if the circular shape of disk was converted to a square.

5.82 THE HASHING ALGORITHM AND HASHING TABLES

In the SM's system, to find a particular frame, its assumed large knowledge base will have to be searched. To assist this, a large table has been constructed which contains the names of all the frames, and their appropriate properties in the knowledge base.

Of course, there are diverse methods of searching this table for a particular frame name and its associated attributes. In a linear search for example, which exhaustively compare every entry in the table with the given keyword, the time consumed is too large and renders this method of searching unsatisfactory.

Binary search, which is more efficient than linear search in particular for a large table, requires an ordered table. The table used in the SM is not and cannot be ordered, as its size is variable and can be changed when a frame is added to or deleted from the knowledge base. Such searching mechanism may therefore have to be used in conjunction with a sort algorithm to order the data. This method, like the one outlined above, will consume a substantial amount of time in the searching process.

It has been suggested (Donovan 83) that ordering tables will not necessarily guarantee a high speed, in particular in situations where large tables are to be searched. In contrast, it is possible to perform faster searching with unordered tables (ie hashing).

As mentioned above, the table in the SM is always subject to changes and modifications. Having to put elements in order will certainly slow down the process. In contrast, a considerable improvement can be achieved by inserting elements in a random way (relatively random).

In the table, the entry-number M (token or key) is generated from the frame's name (keyword), by various methods for address calculation. If the Mth position is available (not occupied), then the new element is put there, if not, then some other place must be calculated for the insertion (clashes).

Hashing is one of the fastest methods for searching (Kanal 1988). It enables the system to locate an entity as fast as the hashing operation takes to generate an entry-number. To generate a number for given keyword (frame-name), we need a deterministic procedure. That is, it will consistently produce the same entry-number of the same keyword.

There are different methods to generate an entry-number from a given keyword. One method is to divide the sum of ASCII values of the characters in the keyword by the

table length, say N, and use the remainder as an entry-number. This algorithm works, as long as there are no common factors between N and the keyword size. Also, if this condition is satisfied, for a group of M keywords, the remainders should be fairly evenly distributed over 0 to N-1 (Donovan 1983).

For example, for four character keywords and a table of length N = 100 say, with the keyword size of 32 bits (one byte or 8 bits * 4) to generate entry-number for a keyword 'car', the following calculation is done :
The ASCII values of characters in the keyword 'car' = (99 97 114)
The generated hash number = remainder = (+ 99 97 114) / 100 = 10

In the SM system a hashing algorithm has been developed to generate a hashing number (entry-number), which can be represented as a token in the hashing table. The algorithm involves a number of steps as follows :

1 - The keyword is passed to the hashing system which decomposes it into its component characters.
2 - The characters in the exploded keyword are translated to their equivalent ascii values.
3 - The ascii numbers are put together in pairs, eg (12 34 17 54) would result : 1234,1754.
4 - All the pairs are added together and divided by the maximum number of entries of the hash table. The remainder of this division is the hashing number of the key word.

There are two different problems that arise from this technique. The first problem becomes relevant when there is not even number of ascii values to constitute the pairs. If such a case arises, the hashing algorithm will add the value of 32, which is the ascii value of the character 'space', to the last number.

The second problem which is more serious than the first one, is that there is always a possibility of hashing clashes. That is, the result of a hashing calculation may be the same for more than one keyword. In the SM's system the hashing algorithm has been developed to cope with this problem. It checks the hash table to see if the calculated address in the table has already been occupied. If the address is taken by another keyword, the hashing mechanism changes the order of the keyword and then passes it back to the algorithm to calculate a new hashing value. This safeguard mechanism guarantees that there will not be more than one keyword with a same hash number, even after the first hashing clash, the mechanism checks the new address to see if it is occupied and will continue until an unoccupied position is allocated to the keyword.

Meanwhile, the mechanism registers any hashing clashes and stores it in a file which can later be used as a reference. At any call for hashing, the system will check that the newly generated entry-number has not been generated previously. If the number has been generated, the system will check the clash-file to find the equivalent re-ordered keyword.

An example using this method to generate a new number for a keyword 'mercedes' :

1 - Exploding the word 'mercedes' which results to
(m e r c e d e s).
2 - Convert each character to its ASCII version :
(109 101 114 99 101 100 101 115).
3 - Putting the ASCII values together in pairs :
(109101 11499 101100 101115)
4 - Add the pairs and the remainder of dividing the result by the maximum number of entries is : 322815/2000 = 815.

The generated hashing value of the keyword 'mercedes' is 815. In the SM the max number of entries is 2000 which can be increased later if required.

The above hashing algorithm has been implemented and adopted by the SM. The biggest advantage of this method over the original one is that there is no word-length restriction on keywords, except those that are imposed by the machine's hardware (ie word-length etc).

In Lisp, hashing, as with other languages, is an efficient way of mapping any Lisp object (a keyword) to an associated object. Hashing tables are provided as primitives of Common Lisp (Steele 84), but in ExperLisp (the dialect which the simulation has been implemented in) this facility is not provided. Therefore, a hashing algorithm was developed.

5.821 STRUCTURE OF HASH TABLES AND THEIR COMPONENTS

The overall controller of the SM embodies two hash tables that contain relevant data which are used for identification, retrieval, mapping of appropriate hierarchies, and communication and message passing purposes. The first table is created at the same time that the knowledge base is created, whereas the second table is created at run time.

The structure of the first table is shown in figure 5.11. This table is called "disk_index" which is used to identify the appropriate frame in the knowledge base,

and find out its position on disk. As mentioned before, in the simulation, a two dimensional array is used to represent this table, and is called "disk_index" that has indices of (2000 * 4).

Hash number	Frame-name	Row-number	Column-number
128	a12	4	23
..	..	..	..
..	..	..	..

Figure 5.11, the structure of the disk_index.

Disk_index contains all the names and the hashing values (keywords and entry-number) of the frames in the knowledge base. The retrieval facilities developed in the simulation, use this table to check the existence of a frame in the knowledge base to retrieve the appropriate hierarchies.

The "controller_copy" table is created at run time (see figure 5.12). It is used in conjunction with disk_index to yield the child/parent relationship, the position on the rectangular array of PEs, and whether the frame in question is a root-node or not.

Parents	Children	Root-node	PE's position
(781 a12)...	(1234 p45)..	none	(3 12)
(182 z41)..	.....	.....	.....
.....	.....	.....	.....

Figure 5.12, the structure of the controller-copy.

Although some of this information is embedded in each particular frame (eg, parent/child relationship), this table contains data that is produced at run time which can readily be used for further mapping, determining the injection point, and communication and message passing. The hashing operation is much faster than the accessing of each frame, whether on disk or in the array of PEs. also in this table, the position of each PE that contains a frame from the mapped hierarchies, is recorded, in conjunction with parent/child attributes and the frame's position on disk (from disk_index). This table is an important facility that is extensively utilised, at run time, by the controller.

5.83 RETRIEVAL AND MAPPING OF APPROPRIATE HIERARCHIES

In the simulation there are certain facilities developed for the identification, retrieval and mapping operations, and these are discussed below.

The controller uses the disk_index to determine whether the object given in the query exists in the knowledge base. This involves calling a top level function called "find_object". Find_object, and its associated functions, calls the hashing algorithm to generate a hash-number for the query-object. The hash-number is then used to check its corresponding entry in the disk_index table to determine the position of the query-object on disk. If the object does not exist in the knowledge base (not specified in the disk_index), the controller will inform the user.

When the position of the query-object on disk is established, the controller performs a series of operations by using the disk_index table to find the root-node of the hierarchy of which the query-object is a member. This process involves going through the hierarchical relationships that are defined in each frame in the knowledge base. That is, when a hash-number for the query-frame is generated, the controller locates the position of that frame on disk via disk_Index and reads its parent/child relationships. In each frame in the knowledge base, the ID_slots, contain the names of its parents and children. These names are coupled with their hash-numbers. The controller can then use these hash numbers to find their associated frames on disks and continues this process until the root-node is found.

When the root-node is found, the controller retrieves the hierarchy from disk and maps it to the array of PEs. The operation involved with retrieval is analogous to lifting a tree from its root and mapping each node, to its corresponding PE. While the retrieval operation is in process, the controller either creates the controller_copy, if it has not already been created (first mapping operation), or it updates it.

In the mapping operation, the controller first checks the controller's copy of disk_index (controller_copy table) to see if this operations is required. The PE's position that contains a mapped frame is registered in the fourth column of that table. If this column is empty, the controller deduces that that frame has not been mapped to the array of PEs.

Note that, in the simulation, further performance of the mapping operation is always a possibility. A frame may require inheritance from other hierarchies that have not been mapped to the array of PEs.

Thus, the mapping operation may occur a number of times. In the simulation, this repetition is recorded by a Free variable called; Free_maphistory. This variable, before the first mapping operation, has 0 value. It gets updated more and less like the loading operation of disk_array. After each mapping operation, an integer which represents the number of frames of the largest level in the hierarchy is added to it so that, in the next operation, the next hierarchy is mapped next to the already mapped hierarchy starting at the position : (0 Free_maphistory). This will ensure that the following hierarchies will not overwrite each other. In figure 5.14, a diagrammatical representation of the rectangular array of PEs is shown. Also, in this figure, there are four hierarchies, with their logical relationships, that have been mapped to the array. The value of the Free_maphistory after the first mapping operation will be 8. This indicates that the next mapping will start from the position of (0 8) and according to the number of frames in each level, the values of appropriate rows will be calculated. This means that, always, the first level to be mapped to the array is the lowest level of the hierarchy.

Having more than one hierarchy mapped to the system, imposes two constraints on the whole operation. First, it has to be ensured that the consequent hierarchy would not overwrite those hierarchies already mapped to the system. Second, how should these hierarchies be mapped to the system so that, the number of redundant PEs will be reduced to minimum ?

By introducing Free_maphistory, the first problem is no longer a threat, but the latter problem that should be considered here. One option is to map each subsequent hierarchy next to the previous hierarchy, as if putting two triangles next to each other to form a square. One benefit gained from this type of mapping will possibly reduce the length of links between two hierarchies. The distance between a parent on one hierarchy and its children on an upside down hierarchy will be reduced by children being positioned nearer than if these tree-like structures were all hanging by their root-nodes. This concept is shown in figure 5.13, where a number of hierarchies are mapped in the manner discussed above.

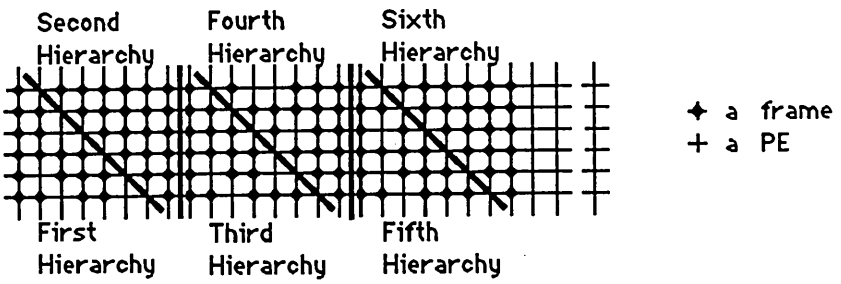


Figure 5.13, mapping of several hierarchies.

The main advantages of this type of mapping, is that the number of redundant PEs will be reduced. On the other hand, the main disadvantage of this type of mapping, is that, the propagation always starts from all the root-nodes of the mapped hierarchies, and to explore each hierarchy in parallel the propagation flow should start from different places in the rectangular array of PEs. This in turn, imposes unlimited complexities on the propagation of messages and on the overall communication.

The mapping method employed for the simulation is to map each hierarchy next to the previously mapped hierarchies with all their root-nodes being positioned at the top (see figure 5.14).

Note that, the mapping scheme suggested above, is by no means regarded as an efficient mapping operation. On the contrary, it is extremely simple, and probably adds to the existing overheads associated with the communication and some redundancy of PEs in the array. There are various methods for implementing mapping operations (Bokhari 1981, Moldovan 1986), but the development of an appropriate mapping mechanism would require considerable attention which can not be provided in the life span of this project.

5.9 RECTANGULAR ARRAY OF PES

In the simulation a two dimensional array represents the rectangular array of PEs with indices of (5 * 40). Note that as to the limited memory size available to the simulation, this array is relatively small but it does serve its prescribed purposes.

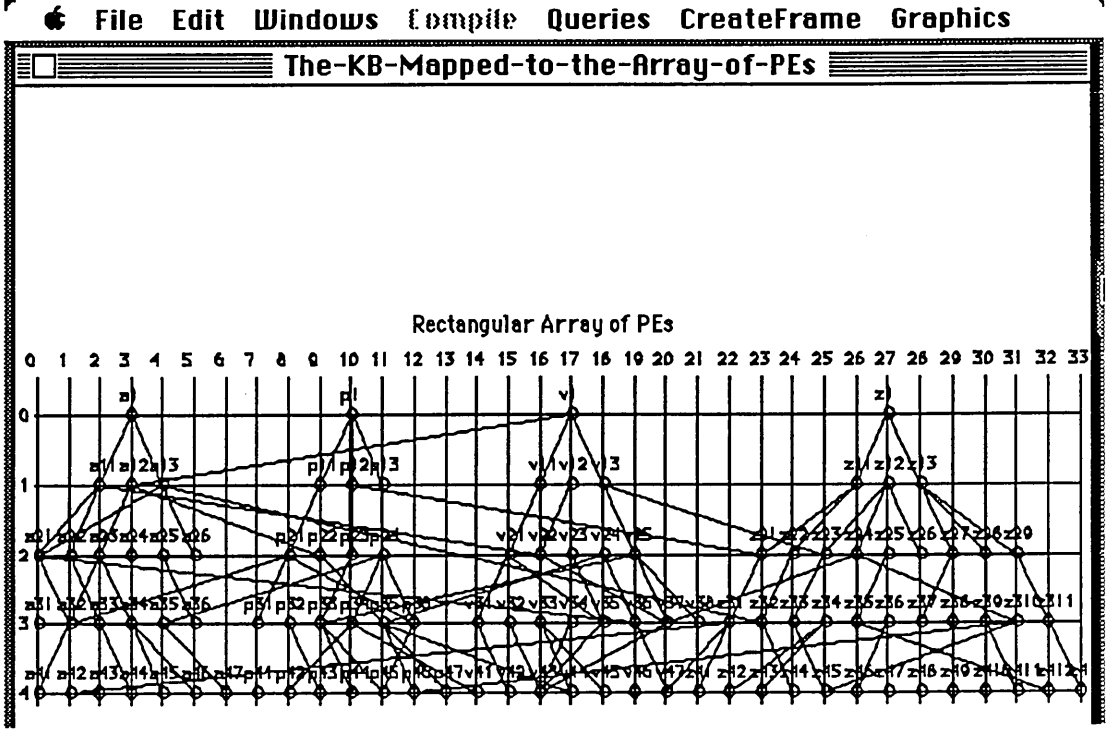


Figure 5.14, a screen dump of graphics representation of a rectangular array of PEs.

In figure 5.14, a screen dump of graphics representation of rectangular array of PEs is shown. This figure shows four mapped hierarchies with their logical relationships.

5.91 PEs, THEIR FUNCTIONS AND CONTENTS

In the simulation program all the PEs are implicitly represented. It is assumed that each PE contains the distributed/replicated inference engine, which performs the appropriate matching and inheritance operation according to the type of a given query and the packet that it receives. Thus, after receiving a packet from its neighbouring PEs, according to the packet's contents, it performs appropriate matching and inheritance operations on the frame that it contains. It stores a copy of updated packet in its memory before passing it to the next PE. Another function of each PE therefore, is to assist in the overall propagation by receiving a packet and passing it to its appropriate neighbour.

At run-time, the contents of an active PE are as follows :

1- A frame mapped from the KB.
2- A packet that contains :
    query's slots/values,
    upward pointers and,
    inherited slots from higher level frames.
3- Time list.

The time list is a list that contains two different sets of numbers, representing the time penalties on knowledge processing and communication. In chapter 6, the time list and its contents will be discussed in detail.

Here, it must be mentioned that the hardware (architecture) of the SM was defined prior to the start of this project. This in turn has imposed certain constraints on the process of defining and developing an appropriate knowledge representation language, and the method adopted for its execution.

6.0 CHAPTER SIX : BENCHMARKING AND EVALUATION

6.1 INTRODUCTION

As discussed in previous chapters, the purpose of building the simulation for the Sheffield Machine ("SM") was to study and analyse its behaviour, which is characterised by the operations involved in knowledge processing and communication. As a result of this evaluation, the advantages and disadvantages of the SM's parallel architecture, and the execution of its dedicated frame-based language, are discussed and determined. The central objective is to ascertain the SM's suitability and practicability as a parallel machine, and where appropriate to recommend modifications.

In the simulation, each Processing Element ("PE"), is capable of performing the following operations :

a) Pattern matching and inheritance operations (knowledge processing).
b) Updating the contents of each packet that it receives.
c) Storing a copy of the packet in its memory
d) Sending the packet down to its appropriate neighbouring PEs (communication).

Knowledge processing and communication constitute the overall operation of the simulation, which provides the core element for its analysis and evaluation. Thus, the simulation's performance is quantified by calculating the time penalties for its two major operations : knowledge processing and communication.

In the following subsections, the algorithm and criteria for time delay, test-runs and their analysis by the graphics package, and the conclusion, are presented.

6.2 TIME DELAYS

The hardware components of the SM consist of a large number of PEs (33 x 33) a lattice of buses, a controller, a parallel disk unit and a switching network (see figure 5.1, chapter 5). In the simulation, the time penalties only for those operations that have direct effect on the SM's performance have been calculated. This includes disk access/transfer, communication, propagation of messages and knowledge processing, inheritance operations and their associated matching operations.

The SM's knowledge base may be stored on one of the commercially available disk sub-systems. In chapter 5, it was mentioned that a parallel disk unit called DIA (Disk In

Array) may be included in the SM system as its secondary storage unit. In DIA a number of magnetic disk drives configured into an array, and run in parallel to achieve a very high access time of 10 mseconds (simultaneous read/write) and transfer rate of 36 Mbytes/s. By including the 10 mseconds time penalty for disk access, the overall access/transfer rate is 35.644 Mbytes/s. This sub-system has a storage capacity of 15 Gbytes, which can be increased to 120 Gbytes by adding extra disk drives to it.

Therefore, it seems reasonable to adopt the time penalties caused by disk access and data transfer operations in DIA, as a criteria for the SM's system. In chapter 4, it was shown that the average size of a frame is approximately 80 bytes. Adding extra communication/identification data to each frame in the knowledge base would probably increase its size to 200 bytes. If we then assume that the average size of each frame in the SM's knowledge base is approximately 200 bytes, the transfer rate between the disk sub-system and the array of PEs would be approximately 180K frames per second. To include the access time, which can be regarded as a fixed value of 10 mseconds, the total rate for access/transfer is 178217.8 frames per second.

Note that the time penalty for frame-transfer through the switching network is not included in the above calculation. This time penalty is relatively small in comparison to disk access/transfer in the SM. For example, the throughput of the Butterfly switch TC2000, with 38 MHz clock speed, is 2.4 Gbytes/s (Trew 1991), which is roughly equal to 12 M frames per second.

By including the time penalty caused by the switching network as above, the overall rate for frame access/transfer between the disk-unit and the rectangular array of PEs of the SM, via the switching network, would be approximately 176K frames per second.

In chapter 3, it was assumed that the SM's rectangular array of PEs consists of 1089 PEs (ie an array of 33 x 33 PEs), where each PE has its own memory of 16 Kbytes. Therefore the maximum number of frames that can be accessed and transferred to the array at any time is 1089 frames, and would take approximately 0.0062 seconds. This calculation has been added to the final result of the simulation's analysis (see section 6.54).

The inter-communication between PEs is based on transferring data packets, where each packet contains query-slots, upward pointers and all the inherited slots from higher level frames. Each packet is passed from one PE to another, and in the process of calculating time penalties for communication the following steps are involved :

a) Compute the size of each packet and,

b) Compute the time taken for each packet travelling from one PE to another PE.

Thus, in order to calculate the time delay for communication, first the size of each packet, and then the distance between its source and destination are calculated. It is assumed that the average time taken for one byte to be sent form one PE to another PE is 100 nsecs[1]. As mentioned in chapter 3, this is the speed of a typical bus to transfer one character (8 bits or one byte).

To calculate the size of each packet, first, the number of characters in the packet are calculated, and then the time penalty for transferring them from source to their destinations, will be calculated.

In the propagation, each frame may inherit properties from its ancestors via the packets that it has received. After the inheritance operation is performed, the contents of each packet is updated and then that packet is passed down to the lower level frames.

The calculation of time penalties for inheritance and matching operations is based on a comparison of two strings. It is assumed that, at the machine level, the first string will be loaded to the accumulator, and the second string will be matched with it, taken directly from the memory. According to the size of the accumulator, each string will be divided into a number of bytes (8 bits). It is assumed that the average time delay for loading 4 bytes to the accumulator is 100 nsecs, and the comparison of the two one-byte characters also takes 100 nsecs (see chapter 3 and the footnote on this page).

In the simulation, various functions have been developed to calculate the time penalties for the loading and comparison of the two strings, or group of strings, and message passing. In unsuccessful matching operations, there will be a time penalty for loading the first byte of the given string.

However, in each interrogation, each relevant PE that is involved in matching operations or transferring packets, will contain a list of time delay penalties. This list is called "time-list" and will be stored in an appropriate file for future use (by the graphics package). Each time-list contains the following information :

((frame-name)

(x/y position of the PE in the rectangular array)

---

1 This is an approximate working figure, derived from an overall comparison of various sources (including Bertsekas 1989, Decegama 1989, Hwang 1987, Hennessy 1990, Ibbett 1989, Morris 1987).

(start-of-transfer end-of-transfer)
(start-of-processing end-of-processing)).

Note that both start-of-transfer and start-of-processing are accumulative. That is, they contain the time penalties of all the previous cycles.

6.21 LOADING AND COMPARISON

As mentioned above, the assumptions made for the loading and comparison operations are as follows :

1 - A character is represented by 1 byte.
2 - The time penalty for loading 4 bytes = 100 nsecs.
3 - The time penalty for comparing 1 byte = 100 nsecs.
4 - At each loading operation, only one byte will be loaded. If the size of each string is more than 1 byte, there will be more than one loading operation.

The operation involves loading one string and matching it with the input string. The loading operation takes the string from a packet as the input string. Note that the result of timedelay for comparison is in microseconds. Note also, the purpose of time delay functions for matching is not to return the result of matching, but to calculate the time taken for their operations. The matching will continue so far as each corresponding character in each string are the same, otherwise, the whole operation will halt and the time taken for matching up to that point will be returned.

6.22 TRANSFERRING DATA PACKETS

In the process of calculating the transfer-time for message passing there are several stages that are as follows :

1- Calculate the distance between the source and destination. That is, the distance between the PE sending a message and the PE that receives it. The distance is measured by the number of PEs situated in between them (DIST).
2- Calculate the number of switches between the source and destination, and also calculate their time delays (TR * TR-T-DELAY).
3- Calculate the size of the packet being transferred (SIZE).

The assumption being made is that the average speed of a typical bus is 100 nsecs/character.

Transfer-time = (DIST * SIZE * 0.1) + (TR * TR-T-DELAY)

0.1 μsecs, is the time taken for a character-transfer between every two PEs. The result being returned is in microseconds.

6.3 VERIFICATION OF TIME-DELAY CALCULATIONS

There are two major objectives in calculating time-delay penalties :

1 - Time delay for processing; that is, time taken for inheritance and matching operation.
2 - Time delay for communication; that is, calculating time taken to send a message from point A to point B in the rectangular array of PEs. The messages are sent in data packets, where there is no imposed size limitation on each packet (bandwidth).

6.31 VERIFICATION OF PROCESSING TIME

The calculation of time delay involved in any matching operation is as follows :

1- Time delay for Loading (4 bytes takes 100 nsecs).
2- Time delay for comparison (one byte, a character, takes 100 nsecs).

A series of functions have been developed to calculate the time taken for loadings, ie if the size of a string is more than 4 bytes (we assume that each character is represented by one byte) there will be more than one loading operation. Thus, to calculate the time delay for a matching operation, first the time delay for loadings will be calculated, and then the time delay for each comparison. If the first string is available and there is no second string, only the loading-time is computed. The first string is the string that is being brought in from the memory, and the second string is the string already in the CPU (accumulator).

The following examples demonstrate the operations discussed above. The high level function match_delay, returns the time delay for loading/matching operations between two given strings.

Note : the first argument is loaded and the second is the input string.

```
(match_delay 'example 'exam)
;(number of loading : 2  - number of comparison : 4 )
;.6 μsecs
```

```
(match_delay 'exam 'example)
;(number of loading : 1  - number of comparison : 4 )
;.5 µsecs

(match_delay 'example202000 'exam)
;(number of loading : 4  - number of comparison : 4 )
;.8 µsecs

(match_delay 'example202000 nil)
;.4 µsecs

(match_delay nil 'exam)
;0

(match_delay 'example202000 'example202000)
;(number of loading : 4  - number of comparison : 13 )
;1.7 µsecs

(match_delay 'one 'onetwothree)
;(number of loading : 1  - number of comparison : 3 )
;.4 µsecs

(match_delay 'one 'otwoemnerem)
;(number of loading : 1  - number of comparison : 1 )
;.2 µsecs

(match_delay 'onetwothreefour 'a_byte)
;(number of loading : 4  - number of comparison : 0 )
;.4 µsecs
```

MATCHING STRINGS IN TWO GIVEN LISTS

```
(do_matchdelay '(first-string second-string third-string) '(first-item second-item
last-item))
;(number of loading : 3  - number of comparison : 6 )
;(number of loading : 4  - number of comparison : 7 )
;(number of loading : 3  - number of comparison : 0 )
;2.3
```

This function is used to determine the processing time taken to load and compare the contents of a given list. The given list may contain 'n' number of strings, which are being measured recursively.

6.32 VERIFICATION OF CALCULATING PACKET's SIZE AND ITs TRANSFER-TIME

A packet is a complex list that contains a number of sub-lists. Its structure is as follows :

(QUERY-SLOTS UPWARD-POINTERS INHERITED-SLOTS)

An example of a packet :

```
( nil                                          ;query-slots
  ( ((1749   1 ) a1_inh_slot1)                 ;upward pointers
      ((1749   2 ) a1_inh_slot2)
      ((1749   3 ) a1_inh_slot3)
      ((1749   4 ) a1_inh_slot4)
      ((1849   1 ) v1_inh_slot1)
      ((781   1 ) a12_inh_slot1)   )
  ( (a1_inh_slot1(value  a1_v1))               ;inherited slots
      (a1_inh_slot2  (value  a1_v2))
      (a1_inh_slot3  (value  a1_v3))
      (a1_inh_slot4  (value  a1_v4)) )   )
```

The size of this packet can then be calculated by calling packet_size :

```
(packet_size '(nil (((1749   1 ) a1_inh_slot1)((1749   2 )a1_inh_slot2)((1749
3 ) a1_inh_slot3)((1749   4 ) a1_inh_slot4)((1849   1 ) v1_inh_slot1)((781   1
)  a12_inh_slot1))((a1_inh_slot1(value  a1_v1))(a1_inh_slot2  (value
a1_v2))(a1_inh_slot3  (value  a1_v3))(a1_inh_slot4  (value  a1_v4)))))
;190
```

This means that there are 190 characters in the given packet.
Note that in the above packet, the first element is nil. This is due to the type of query (query-object number 1), where there are no query slots.

The function find_distance1 takes the hash numbers of two PEs (hash number of two frames contained in them) and pass as their x-y coordinates to find_distance, which will return the distance.

eg:

```
(find_distance1  983  981)
;12
```

12 is the number of PEs that exist between two PEs (including) that contain two frames with hashing numbers; 983 and 981 respectively.

The function transfer_time calculates the transfer-time taken for a packet from one PE to another PE. Packet-size is the number of characters in a given packet; distance is the number of PEs situated between two given PEs.

```
(transfer_time  192  12)
;230.4   ;The value being returned is in Microseconds.
```

This means that, the time taken to transfer a packet of size 192 characters from a PE to another one with the distance 12 is 230.4 μsecs.

The function longest_transfer takes a frame and checks all the parents and their distances. It returns the longest transfer_time ie the the transfer time of the furthest parent.

eg:

```
(longest_transfer  '(781  a12))
;115.2
```

6.33 A TEST-RUN EXAMPLE

The query is to find the full detail of a frame called 'a12' in the benchmark knowledge base. The injection points, after mapping appropriate hierarchies, are at PEs containing frames; 'v1' and 'a1'.The propagation starts form these frames and they converge at a PE that contains the frame 'a12'. From the injection points to the target frame (ie 'v1' and 'a1' to 'a12' respectively), the operations involve inheritance of properties, and message passing, which have caused certain time penalties to be incurred. These time penalties are calculated according to the procedures given above. At each cycle, the time penalties so far are stored in the appropriate PE. As mentioned above, the data in each PE is represented as follows :

```
((frame)(packet)(time_list))
```

The complete result of the execution is stored in a file shown below :

```
(v1  (0   9 )(0   0 )(0   .8 ))
(a1  (0   0 )(0   0 )(0   2.8 ))
(a12  (1   1 )(0   115.2 )(2.8   35.1 ))
```

Each item in each list represents the following concepts :

1- The frame's name.

2- The position of the PE in the rectangular array of PEs that contains that frame.

3- The transfer time taken for the packet to arrive at that PE.

4- The processing time taken for performing appropriate inheritance operations.

Note that in the first two lists :

(v1 (0 9)(0 0)(0 .8))

(a1 (0 0)(0 0)(0 2.8))

both frames were identified as the injection points, and there are no time penalties for transfer (ie these are the points were packets were injected), and yet a certain amount of inheritance operations were performed (0.8 μsecs for frame 'v1' and 2.8 μsecs for frame 'a1').

However, in the third list given above, there are time penalties for both transferring and processing operations. A packet has arrived at a PE that contains 'a12' (transfer time), and a certain amount of inheritance operations have been performed on 'a12', which are shown in the following list :

(a12 (1 1)(0 115.2)(2.8 35.1)).

Which means that, it has taken 115.2 μsecs for the packet to arrive at this PE from a12's parent. Also, it has taken 32.3 μsecs to perform inheritance.

Note that in this example the frame 'a12' has more than one parent; 'a1' and 'v1' of which 'v1' is the furthest parent hence, the time penalty for data packet transfer from 'v1' is :

```
(longest_transfer '(781 a12))
;115.2 μsecs.
```

The contents of the PE holding 'v1 :

The frame itself

```
(((1849    v1)
  (id_slots ((offspring_no (value 3 ))
             (level_no (value 1 ))
             (parents (value none))
                  (children (value ((967  v11)(1967  v12)(981  v13)(781 a12))))))
   (ind_slots ((v1_ind_slot1 (value v1_ind_v1))))
```

(inh_slots ((((1849 1) v1_inh_slot1) (value v1_inh_v1)(inh_condition default)))))

The packet

```
(nil   ; Nil is for absent query-slots.
 (((1749   1 ) a1_inh_slot1)
  ((1749   2 ) a1_inh_slot2)
  ((1749   3 ) a1_inh_slot3)
  ((1749   4 ) a1_inh_slot4)
  ((1849   1 ) v1_inh_slot1)
  ((781   1 ) a12_inh_slot1)) ;These are upward pointers from 'a12'
 ((v1_inh_slot1 (value v1_inh_v1)))) ;This is the inherited slot in the packet.
```

Time-list

```
(v1 (0   9 )(0   0 )(0   .8 )))
```

Thus, the contents of each PE is :

(Frame packet time-list)

Here, in the PE containing 'v1', the packet's size is :

(packet_size '(nil (((1749 1) a1_inh_slot1)((1749 2) a1_inh_slot2)((1749 3) a1_inh_slot3)((1749 4) a1_inh_slot4)((1849 1) v1_inh_slot1)((781 1) a12_inh_slot1))((v1_inh_slot1 (value v1_inh_v1)))))

;128

The distance between the PE containing 'v1' and the PE containing 'a12' is :

781 is the hashing value for 'a12' and

1849 is the hashing value for 'v1'.

(find_distance1 781 1849)

;9

The transfer time for transferring the packet from 'v1' to 'a12' is :

(transfer_time 128 9)

;115.2

Where 128 is the packet size and 9 is the distance between PEs containing 'a12' and 'v1' respectively.

In order to examine the correctness of the calculation of time penalties for knowledge processing, let us check a simple case; 'v1' frame. In the PE containing this frame,

there is a packet which has been created at injection time. This packet is given above, but we are more interested in the second part of it (see above), which is the inherited slots (or to-be inherited slots) :

```
(v1_inh_slot1  (value  v1_inh_v1))
```
and the match delay for this list is :
```
(do_matchdelay  '(v1_inh_slot1  (value  v1_inh_v1))  nil)
```
;.8 μsecs.

Time penalties produced at run time for both processing and transferring are the same as those produced in this example, and therefore the procedures used for calculating time penalties are operating satisfactorily and are correct.

6.4 GRAPHICS PACKAGE

In order to show the prominent behaviour of the SM in a more elaborate representation, and also to simplify the analysis of its behaviour, a graphics package was developed as part of the simulation. This package produces three[3] graphs for each test-run. The first graph, produced at run time, draws the rectangular array of PEs, the mapped hierarchies and all the relevant propagation paths, by utilising data produced during the execution. The graphs numbered 1/2 and number 5, use the information produced at run time and stored in a file. These two graphs are used for analysis and evaluation purposes, and show different perspectives of the machine's activities.

As mentioned above, the information required to plot these graphs are created at run time and stored in a file. The structure of each file is as follows :

```
( ((Fn (Xn Yn) (Tn Tn+1) (Pn Pn+1))
  ((Fn+1  (Xn+1  Yn+1)  (Tn+1   Tn+i)(Pn+1  Pn+i))
  .
  .
  .
  ((Fn+m  (Xn+m  Yn+m)  (Tn+m  Tn+m+i)  (Pn+m  Pn+m+i))
 )
```
In this file each list consists of four elements; frame name (Fn), the (x y) coordinates of the PE which contains the frame Fn, the start and finish of transmission time (Ti), and the start and finish of processing time (Pi).

---

[3] In fact there are seven different graphs that can be produced, but in this thesis because of the lack of space, only 3 are given.

This information will then be fed to the graphics package which will draw the required graph. In the following sub-sections, the nature of each graph will be discussed in terms of its representation, its relation with relevant queries, and the possible conclusions that may be drawn from it.

6.41 GRAPH NUMBER ONE

The computational model of the SM is based on downward transitive closure of messages through the paths which are provided by the relationships between frames. Along these paths, the inheritance of properties are performed. In any interrogation, although all the paths are being traversed in parallel, only a few of them carry the actual information relevant to the query. In graph number one these relevant propagation paths, the rectangular array of PEs, and mapped hierarchies are shown.

In figure 6.1, graph number one is shown. It can be seen that all the necessary information to study a test run is given including the visualisation of the propagation paths, injection points and every frame involved in the test run. In addition, in this graph an option is available to draw the relationships between all the mapped frames in the array of PEs.

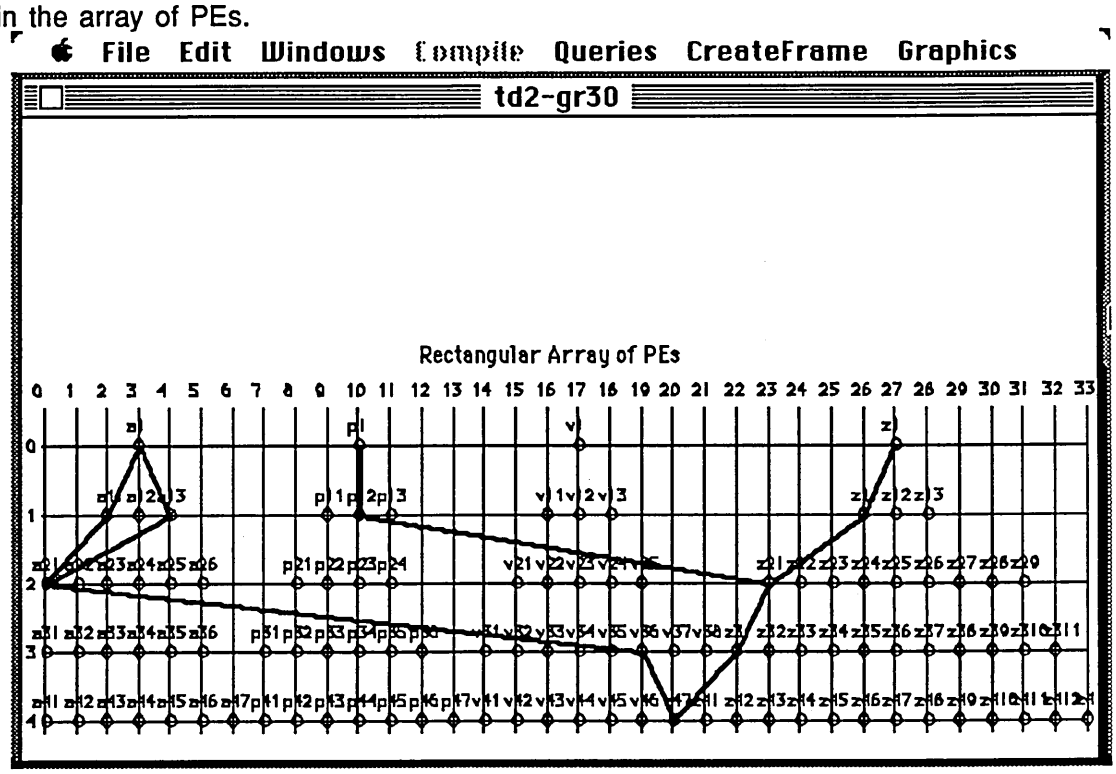


Figure 6.1, graph number one.

6.42 GRAPH NUMBER 1/2

In this graph the time penalties for processing all the query-related frames and for their inter-communication, are presented (see figure 6.2).

Graph number 1/2 provides a clear comparison between the time taken for query-related frame-processing and their communication. It also shows the number of PEs containing these frames for each time interval. In each execution the increase and decrease of the number of active PEs can be shown in relation to different time slots. In figure 6.2, the time penalty for frame-processing is shown by a thin black line which ends at the point marked "P". The time penalty for communication is shown by a thick black line which ends at the point marked T.

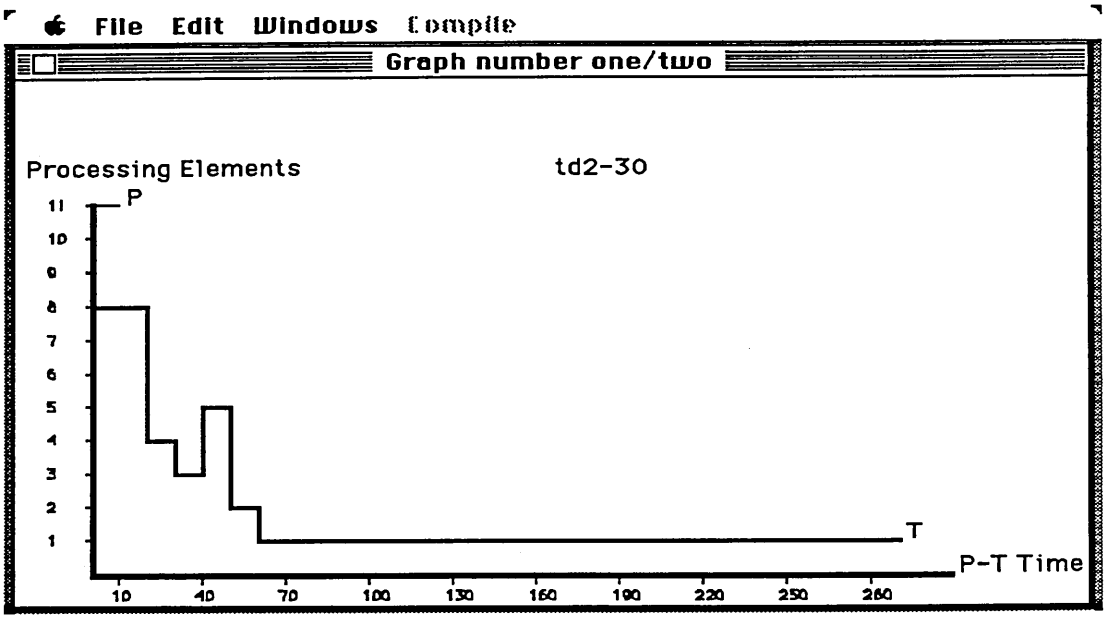


Figure 6.2, graph number 1/2.

In this figure it can be seen that the time taken for frame-processing and communication has taken approximately 15 microseconds and 200 microseconds respectively. It can also be seen that at the time interval 10 - 40 microseconds, the number of active PEs communicating with each other has dropped from 8 to 3, and then up to 5.

6.43 GRAPH NUMBER 5

In this graph, serial and parallel execution of the same query is represented and compared. Graph number five is a combination of two graphs, one superimposed on the other (see figure 6.3).

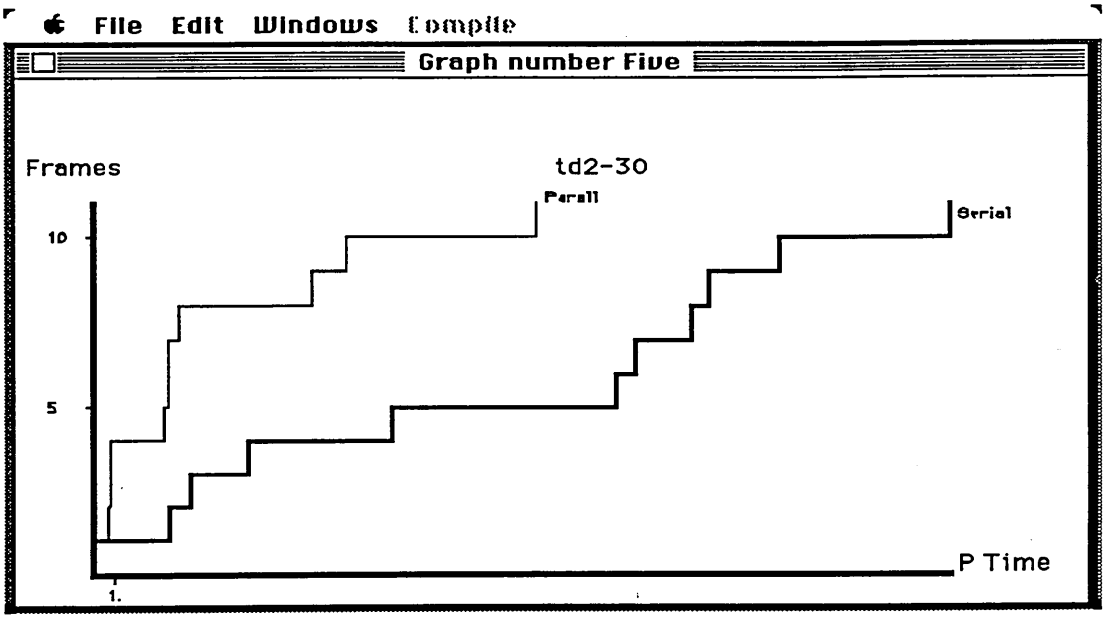


Figure 6.3, graph number 5.

In this graph two curves are drawn; the first curve shows the processing time for each group of query-related frames being executed in parallel. The second curve represents the processing time for serial execution of all the relevant frames. On the x-axis, the time scale for frame-processing is presented, with the y-axis scaled to the number of frames being processed at each run. The main feature of this graph is to compare both parallel and serial processing together on the same graph in order to show the gain in speed.

6.5 EVALUATION AND CONCLUSIONS

In the Artificial Intelligent ("AI") community, people have accepted the fact that parallelism is one of the major factors in the development of faster and more powerful machines. This conclusion, as mentioned in chapter one, is explicitly based on today's requirements and applications. At the same time, with a parallel architecture, a suitable knowledge representation scheme is required by which the relevant information in the application domain will be represented. In an ideal machine therefore, while the search space of an application domain is being explored in parallel, appropriate conclusions are made by embodied inference mechanisms. This exploration is then expected to be extremely fast, which will then be able to cope with the constraints imposed by complex applications.

Taking a simple approach, we can assume that, in order to gain speed and power, many PEs would work together and simultaneously execute similar portions of a given task. Thus, a strategy could be developed to partition a given problem into smaller

problems, so that all or most of these problems will be solved simultaneously, and the combination of their results will constitute the final solution to the original problem.

Partitioning the problem domain raises three important issues :

a) partitioning itself,
b) complexity, and
c) communication.

The computational problems can be divided into smaller portions and explored simultaneously, but at the same time, there are some problems that are inherently serial and can not be partitioned. Adding further to this dilemma, we do not exactly know which problem is serial and which is parallel (Appalaraju 1984, Reeve 1989). This is due to the complexity of the given problem, although it may by possible to discover the nature of the problem, whether serial or parallel, in later stages.

The purpose of this discussion is not to philosophise on the concept of decomposibility of computational problems. In chapters 4 and 5 it was shown that frame-based languages are suitable for the SM's parallel architecture, and the model of it was presented. This type of language facilitates the concept of problem partitioning into smaller portions, by introducing frames. Although frames are powerful and flexible representation models, it is extremely difficult to prove that frames are suitable for all possible applications.

It can safely be assumed that most of the problem domains can be partitioned and solved in a series of mixed parallel/serial operations (Minsky 1980). In a set of processors (or PEs), each can process a particular portion of the decomposed task. To calculate the overall time penalty taken for serial and parallel operations the following assumptions can be made (Krishnamurthy 1989) :

$T_{total} = T_{serial} + T_{parallel}$

Thus, the speed up can be calculated by the following :

Speed-up = T_{total}/T_{serial}

If for example, the serial operation takes 10% of the overall time, the maximum speed-up will be 10, or 50%, the maximum speed-up would be 2.

The third issue is related to communication. That is, when a large problem is decomposed into smaller problems, frames for example, it may be necessary to provide a coordination between the tasks which require communication (ie inheritance and data packet propagation). This can be highlighted by the fact that, the larger the

number of PEs, the more communication links are required. Each PE, for example, may be required to communicate with, say, its 4 immediate neighbours. In a rectangular array of 1000 PEs, at least 4000 communication lines are required. The problem here is that, in addition to the time taken for processing information, in a parallel system we also need to calculate the time taken for communication. This concept introduces a new dimension to both complexity and the design of parallel systems.

In the design and development of parallel machines, from fine grain to coarse grain architectures, the communication cost is one of the important constraints. In a fine grain parallel machine (eg, neural networks, NETL, etc) the communication is based on sending simple, and small, signals. At the other end of the spectrum; in coarse grain design, the communication is based on a large amount of data carried by structures like data packets.

In the simulation of the SM, its frame-based knowledge representation language deals with partitioning, and decomposibility, which was discussed in chapters 4 and 5. The remaining important issue is to compute and examine the time penalties for processing frames and their inter-communications.

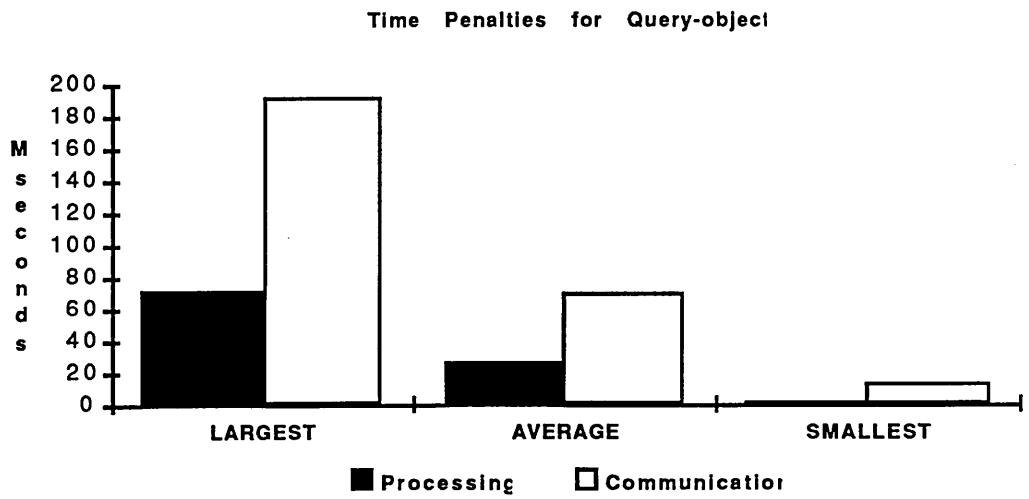
In the course of examining the SM system, a series of tests have been carried out. These tests are performed by means of querying the system, where in each query, the time penalties for both knowledge processing and communication are calculated and stored in a file.
In the SM simulation there are 4 types of queries by which the system was examined. Appendix D contains some of the tests, accompanied by their appropriate graphs. For each test, the query, time-list, a graph representing the successful propagations at run time (graph number 1), graph number 1/2 and graph number 5, are given.

In the following subsections, from studying test-runs appropriate to each type of query, certain charts have been produced in which the overall estimated gain in speed-up is presented. It is important to note that the charts presented in the following sections represent the data produced on processing only the relevant frames to given queries.

6.51 QUERY-OBJECT NUMBER 1'S TEST-RUNS

In query-object number one, the complete definition of a frame, whose name is given, is required. The test-runs show that the amount of time taken for communication in this type of query is one of the highest. Figures 6.4 shows a range of maximum,

average and minimum time for frame-processing and communication of all the test runs in query-object number one.



Query-object1	Processing	Communication
LARGEST	70.73	190.85
AVERAGE	25.4	68.9
SMALLEST	0.8	12.48

Figure 6.4.

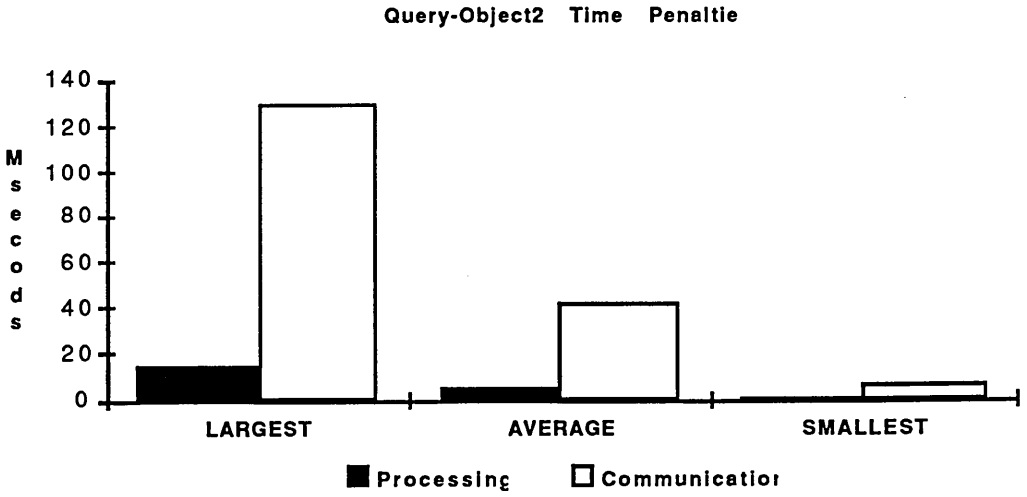
The excessive time penalties for communication is due to the large amount of slots/values, that have been continuously processed and then passed down the appropriate propagation paths to lower level frames. In this type of query, the matching and inheritance operations involve all the slots of each relevant frame. After the appropriate manipulations are done, the updated data packets are moved down the propagation paths, and the size of each packet, and thus time penalties for communication, increase very rapidly.

In appendix D, some of the test runs for this type of query are presented. At the start, the queries were made about objects at higher levels of various hierarchies from the benchmark knowledge base. The time penalties for processing and communication (graph 1/2) were relatively small, and the comparison of parallel/serial processing of relevant frames showed that there is almost no gain on speed (graph 5). By querying objects at lower levels in the mapped hierarchies, the number of processed frames was increased. As a result, the parallel/serial comparison (graph 5) showed an average ratio of 2 to 1 in favour of parallel processing of the relevant frames (ie frames situated on the propagation paths shown in graph No 5).
The relationship between processing and communication in each test run, showed by graph 1/2, indicates a great diversity where, in certain cases the time taken for

processing was greater than that of the communication. This is directly due to the number of slots in each frame, which varies from one frame to another.

6.52 QUERY-OBJECT NUMBER 2'S TEST-RUNS

In query-object number 2, a subset of slots/values of a frame is given (with the frame's name), and as a result of appropriate matching and inheritance operations, the missing parts of those slots/values will be completed. Therefore, the time penalty for communication tend to be less than of that in query-object number one. This notion is clearly demonstrated in the graphical representation of the results (graph 1/2), in appendix D.



Query-object2	Processing	Communication
LARGEST	14.63	128.37
AVERAGE	5.69	41.68
SMALLEST	0.52	6.25

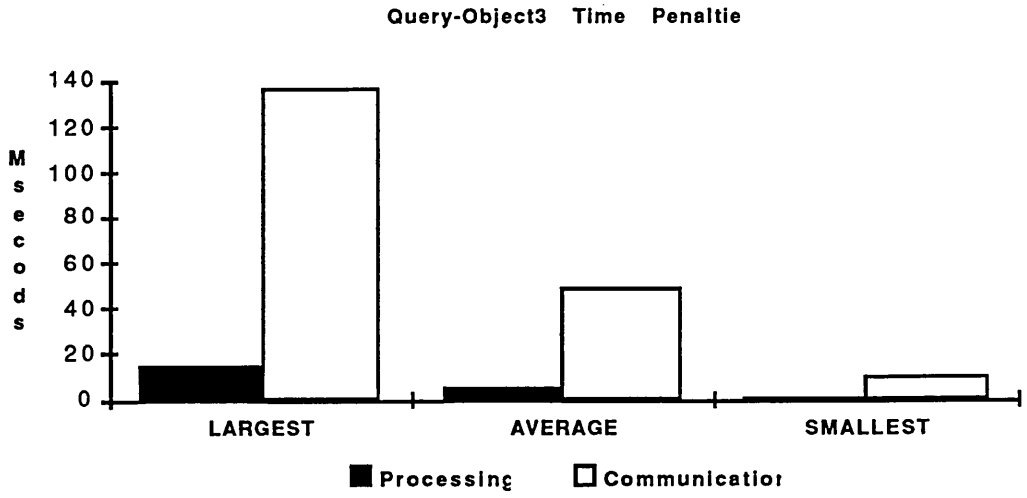
Figure 6.5.

In this type of query, by interaction between the user and the system, a query-frame is made. Packets containing this frame are then propagated through appropriate paths (shown in graph 5). At each level, a check is made to see if the inheritance operation is required. Figure 6.5 shows the largest, the smallest and the average time penalties for both processing and communication, of all the test runs for query-object number 2.

The test runs for querying objects at the lower levels of the mapped hierarchies show that the gap between processing and communication gets larger (graph 1/2). But the ratio between parallel/serial execution (graph 5) of query-related frames stays very close, in spite of the hierarchical position of those frames being queried.

6.53 QUERY-OBJECT NUMBER 3'S TEST-RUNS

In query-object number 3, a full or a subset of slots/values of a frame is given, asking for its name. In this type of query, after the query-frame is made, the user is involved in further interaction to provide the names of appropriate hierarchies. After the mapping operation, the propagation of data packets starts. At each level (or cycle), the inheritance operation is performed on each receiving frame, then the matching operation between the slots/slot-values of the query-frame and the receiving frame is performed. If the match is successful, the frame name is returned to the user.



Query-object3	Processing	Communication
LARGEST	18.23	135.87
AVERAGE	8.84	48.06
SMALLEST	1.39	9.67

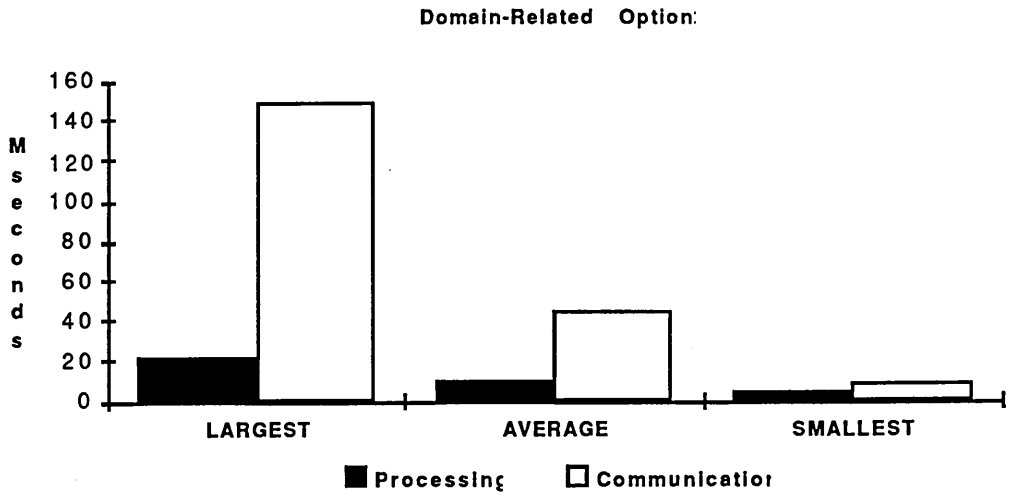
Figure 6.6.

The operations involved with this type of query are similar to that of query-object number 2. Therefore, the time penalty for communication in query-object number 3 tends to be less than in query-object number one. Figure 6.6 shows the largest, the smallest, and the average time penalties, for both processing and communication of all the test runs for query-object number 3.

In query-object number 3, the user provides the same information to that of query-object number 2, but without the frame-name. The inheritance operations in both types of query are the same. In both queries, at each run, the contents of each frame in the path of propagation are examined to check whether or not to perform inheritance operations. All the slots of the query-frame are matched against the slots of the receiving frame, and if the match is successful, the receiving frame's name will be returned to the user.

6.54 TEST-RUNS FOR DOMAIN RELATED QUERIES OPTION TWO

This type of query enables the user to ask for a wide range of information about the relationships amongst frames in hierarchies, in domains, and in the knowledge base as a whole. The user can, for example, provide one or more object-names plus one or more slots with (or without) their values, and variables representing unknown shared slots or slot-values. In the course of propagation, these variables will be bound to appropriate values.



Query-domain2	Processing	Communication
LARGEST	21.59	149.25
AVERAGE	10.84	44.45
SMALLEST	3.68	8.72

Figure 6.7.

In this type of query, the propagation starts from the injection points (the same as every other query available in the SM). While the packets are moving down from higher level frames to their children, at each level (or cycle) the inheritance and matching operations are performed. After the inheritance operation on each frame, by successful matching between its slot-names and slot-values with those in the packet, the variables representing the absent frame-name, slot-name or values will be bound to appropriate values. This process will continue until the leaf-nodes are reached.

Figure 6.7 shows the largest, the smallest, and the average of overall time penalties for both processing and communication respectively.

6.55 CONCLUSIONS

The simulation's test runs have produced a substantial amount of numerical data, which represents the time penalties for processing/communication of those frames

that are explicitly related to the given queries. In the previous sub-sections for each type of query, the result of test runs, accompanied with appropriate graphs, were discussed.

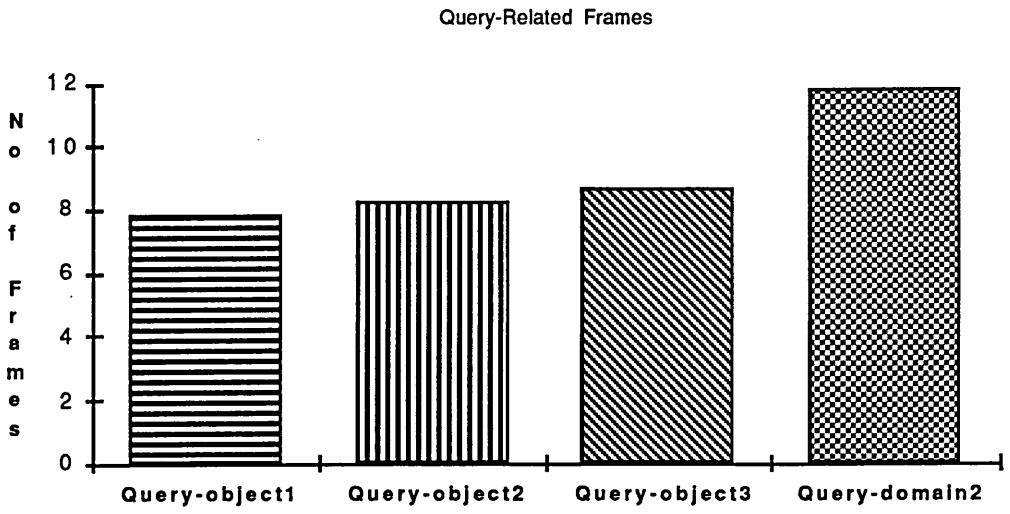
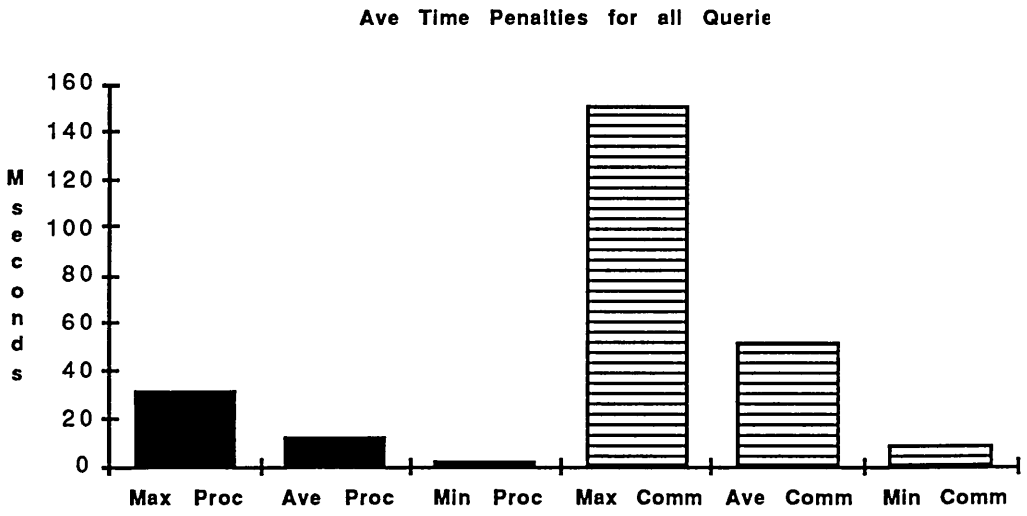


Figure 6.8, the average number of query-related frames in 4 different queries.

Further analysis of test run results is shown in figure 6.8, which presents the average number of processed-frames for each type of query. From this figure, it can also be shown that the overall average number of processed-frames (query-related frames) is approximately 10 frames per run. That is, it would involve an average of 10 frames from which the relevant information can be extracted to satisfy a given query.

In figure 6.9 on the other hand, a range of average time penalties for both frame-processing and communication is shown. For example, the average of all the queries maximum processing time is 31.3 μseconds. As in figure 6.8, the information given is only for query-related frames.



FOR ALL QUERIES	
Max Proc	31.3
Ave Proc	12.7
Min Proc	1.6
Max Comm	151.1
Ave Comm	50.77
Min Comm	9.17

Figure 6.9, a range of average time penalties for communication/processing on query-related frames.

Although in each run the number of query-related frames may be as small as 10, the actual number of frames that are being processed at the same time may be as many as 1089 (the number of PEs in the SM). This is due to the underlying computational model of the SM.

The computational model of the SM is based on cyclic downward transitive closure of messages. These messages are traversed, in parallel, through the paths that are defined by a frame's hierarchical relationships. Every path begins from the root-node and repeatedly links a parent frame to one of its children, down each hierarchy, until a leaf-node is reached. To perform inheritance, while a packet is moving down, each frame on its path may add, delete, or collect information to or from it. Although the propagation of messages is not synchronised, their traversal can be divided into a number of cycles. Each cycle would then correspond to a level in the mapped hierarchies.

This enables the SM's system to process a substantial number of frames in a few cycles. It is important to note that the time taken for processing query-related frames and their communications would represent the longest cycles in any execution. During these cycles all the other frames, related or unrelated, have been executed in parallel.

At the early stages of frame processing, preliminary matching operations are involved. If any one of these operations fails, the processing will be discontinued and the time penalty for that process will be calculated. The time taken for unsuccessful matching operations is small, and it certainly takes less time for a successful matching operation. The time penalties for communication amongst unrelated frames will also be very small. That is, when a packet is inserted at the root-node and starts moving down each path, because of unsuccessful matching operation its contents will not be modified and updated. Thus, the time penalty for sending a similar packet from any one PE to its corresponding neighbouring PE in each path would be almost the same (unlike successful frame-processing, where the size of data packets increases with time).

Thus, the time penalties on operations (both processing and communication) involving frames that are unrelated to the query, will be less than those operations performed on query-related frames. As all these operations are performed in cycles and almost at the same time, the time taken for processing frames that are related to the given query will subsume all the other time penalties incurred by unrelated frames.

Note that this computation is explicitly based on the results produced by all the test-runs (see figures 6.4 - 6.7). Note also that these time penalties were focused on processing query-related frames (average of 10 query-related frames, see figure 6.8). From this set of data the overall average time penalties for maximum, average, and minimum of frame-processing and communication were produced (see figure 6.9).

To assess the SM's overall performance, therefore, the time penalties on operations performed by every PE in the rectangular array that contains a frame, are included, whether or not they are relevant to the given query.

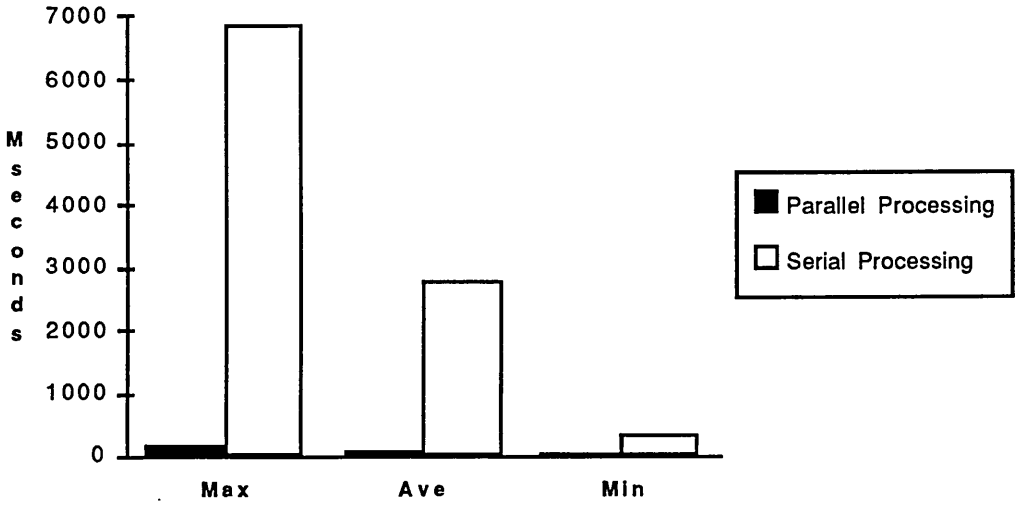


Figure 6.10, a comparison of serial/parallel processing of the same number of frames.

Figure 6.10 shows the comparison between the time penalties for the parallel and serial processing of the maximum number of frames that the SM can process (1089 frames). Note that the total time penalty for parallel execution is the aggregate of time penalties for both processing and communication. In serial processing, on the other hand, only the time penalties for frame-processing are considered.

In figure 6.10, a range of time penalties is presented that includes the average, the maximum, and the minimum amount of time penalties for parallel processing 1089 frames. The maximum time taken to serially process 1089 frames, for example, is

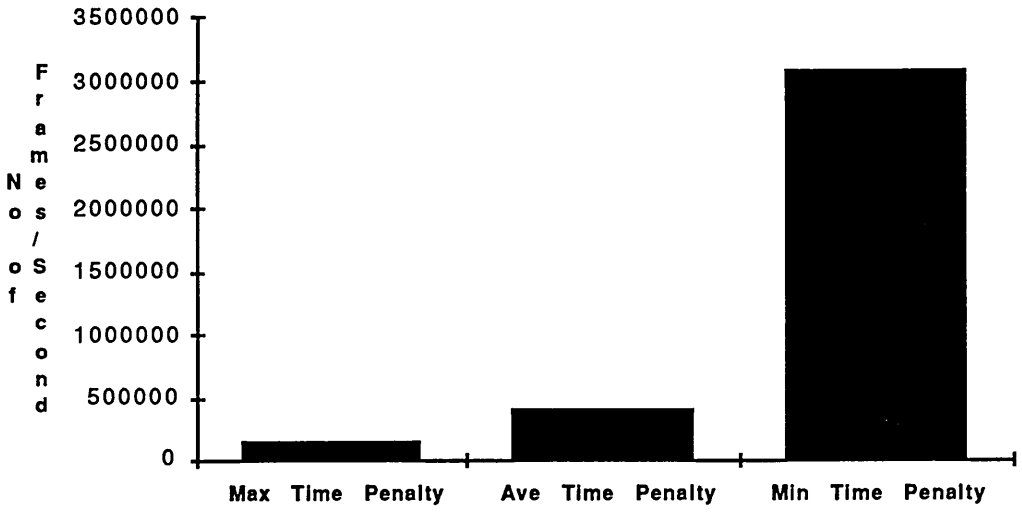
estimated as 6817.14 μsecs[2]. The maximum time taken for parallel processing the same number of frames is 182.4 μsecs (this includes the aggregate of maximum time penalties for frame-processing and communication of the longest cycle).

COMBINED PROCESSING AND COMMUNICATIONS		
for 1089 frames	Parallel Processing	Serial Processing
Max	182.4	6817.14
Ave	63.47	2766.06
Min	10.77	348.48

Figure 6.11.

Figure 6.11 represents the corresponding values for the graph shown in figure 6.10.

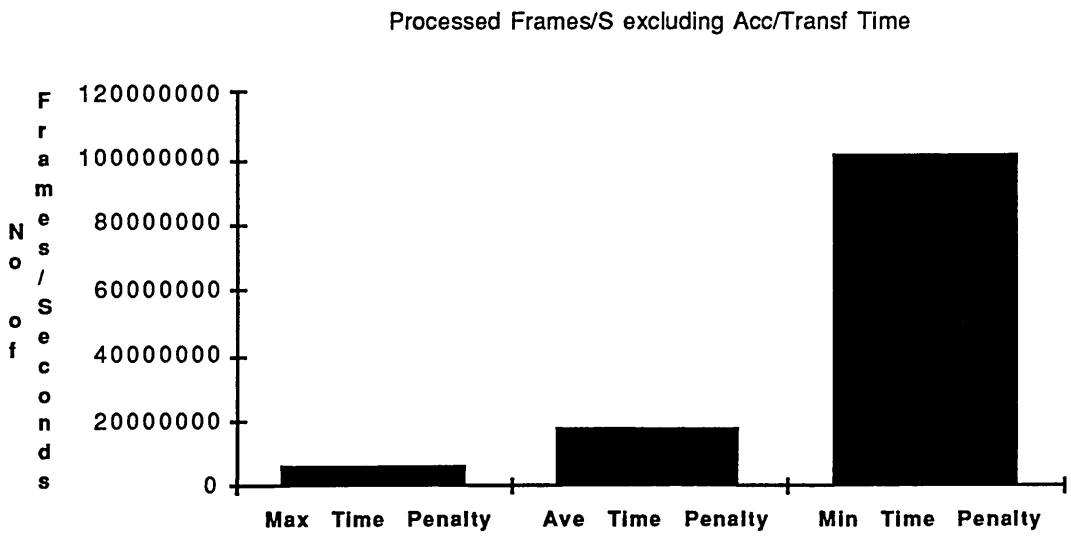
Traditionally, computer performance is measured by determining the number of instructions or floating point operations that can be executed per second. In chapter 3, an attempt was made to determine approximately the SM's performance in this respect. But, as a result of analysing the test runs, new data was produced which can be used as new criteria against which the SM's performance can be assessed. This criteria is based on the performance efficiency involving frames, their retrieval and manipulation. That is, we can measure the SM's performance by the number of frames it can retrieve and process per second.



No of Frames serially processed per second

---

2 As an example: the average of maximum time penalties for frame-processing in all the queries is 31.3 Mseconds (see figure 6.9). We can deduce that the average of maximum time taken for processing each frame is 6.25 Mseconds (in the benchmark knowledge base, there are 5 levels, thus : 31.3/5). From this figure, we can further deduce that the approximate time penalty for 1089 frames is : 1089 x 6.25msconds = 6817.14 mseconds



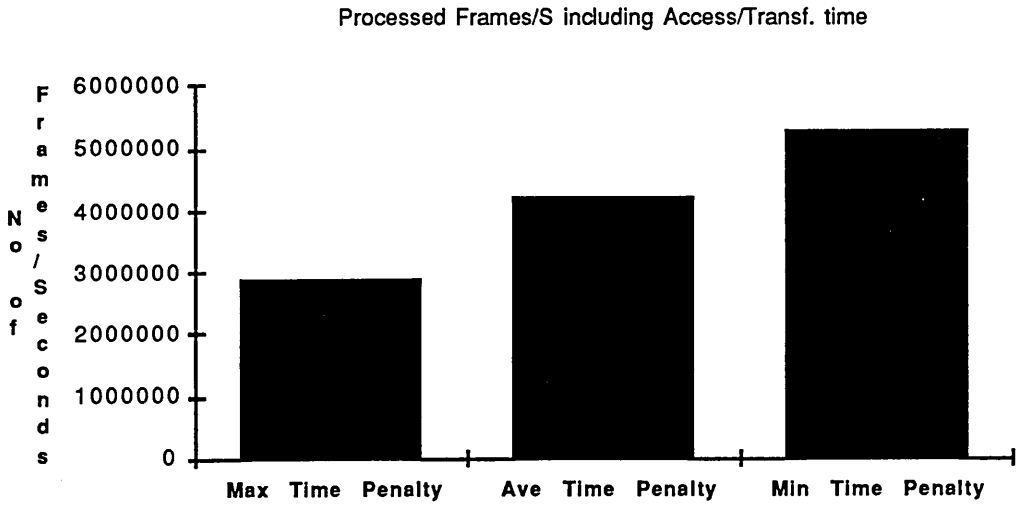
Frames/Sec without Disk Access/Transfer		
	Parallel	Serial
Max Time Penalty	6000000	160000
Ave Time Penalty	17000000	400000
Min Time Penalty	101000000	3100000

Figure 6.12

The graphs shown in figure 6.12 provide a spectrum in which all the possibilities for parallel/serial frame-processing are presented. In the first graph the serial frame-processing is shown, in which between 160K - 3.1M frames may be processed per second. Note that this estimation is based on the amount of time taken to process a frame. In the case of 3.1M frames/S for example, as was shown in figure 6.9, the minimum time taken for frame-processing is very small (approximately 300 nseconds to process a frame). The main reason for this is that the frame concerned was not processed completely. That is, the initial matching operations were unsuccessful; ie as soon as the first character is matched and failed, the process was stopped. This corresponds quite neatly to the assumption that has been made on the time taken to compare a character, which is discussed in section 6.21. On the other hand, the maximum time penalty for serially processing a frame is 6.25 μseconds, which by any standard is too large.

In producing the second graph in figure 6.12, similar assumptions to that of the first graph have been made. This graph shows a spectrum of a number of frames being processed per second in parallel. There are between 6M - 101M frames that can be processed per second. As mentioned above, a large number of these frames are not fully processed, which implies more frames are processed in parallel. Nevertheless, this is certainly a promising result, in particular in comparison with serial performance.

In figure 6.13, the overall performance of the SM is shown, with the time penalty for disk access/transfer is included (see section 5.81, chapter 5).



Frames/Sec with Disk Access/Transfer	
Max Time Penalty	2900000
Ave Time Penalty	4200000
Min Time Penalty	5300000

Figure 6.13.

By including the time penalty for disk access/transfer, a more realistic result has been produced which shows the consideration of other operations outside the run-time execution. Firstly, the result demonstrates the bottle-neck characteristics of the disk access/transfer operations which has reduced the SM's performance at run-time execution by average of 10 fold. Nevertheless, we can always claim that the same time penalty applies to that of serial machines, where the knowledge base is stored on secondary media.

However, the figures and charts presented in this section show a series of important and optimistic points. Firstly, it is obvious that there is a gain in speed-up and ideally the target of run-time execution should be that of 101m frames/second[3].

Secondly, to achieve the minimum time penalties for execution, it is absolutely necessary to reduce the amount of communication. This is shown clearly in every test-run in appendix D and every related graph shown above. One of the most effective ways

---

3 The Intel's i860 is a 40 MHz RISC based CPU with a peak floating point performance of 40-80 MFLOPS (manufacturer's figures), may bring a nice contrast with that of the SM, with the peak performance of 6 - 101 MFrames/S.

of reducing the communication overhead is by reducing the amount of inheritance. This can be achieved by performing inheritance operations at assert time, that is, most of the inheritance should be done while creating the knowledge base.

Thirdly, the method of propagation is another important feature which can play a vital role in achieving high performance. As discussed in chapters 4 and 5, the underlying computational model of the SM has played an extremely important role in achieving high performance and exploiting potential parallelism.

Without the dedicated frame-based language, it would have not been possible to implement a suitable model of computation. The SM's language provides a rich and flexible environment in which both the knowledge and the control mechanism can be fully represented.

7.0 CHAPTER SEVEN : CONCLUSIONS AND FUTURE WORK

7.1 CONCLUSIONS

The central objective of the project was to study and investigate the feasibility of gaining a high speed-up in the parallel execution of AI based applications. To achieve this objective, a new simulation was developed with its own dedicated frame-based language[1]. In the process of executing this language, dedicated software computed the time penalties on different operations performed by the simulation's major components. The numerical data produced at each execution were converted into visual representations, and the analysis of these representations yielded encouraging results.

The architecture of the simulated SM is of a medium grained MIMD type parallel machine, and may be classified from an AI view point as a knowledge based machine of type associative model. It comprises a controller of a conventional type, a (33 x 33) array of PEs, a high bandwidth switching network and a parallel disk unit. The maximum number of PEs in the array is assumed to be 1089, where each PE, via tri-state drivers, is connected to a lattice of horizontal and vertical buses. Each PE is also assumed to be a 32-bit microprocessor type, with 33 MHz clock speed, and 16 Kbyte of memory, thus constituting a 17 Mbytes aggregate memory for the SM. The performance of each PE, with 33 MHz clock speed, is assumed to be 4 Mips, resulting in an aggregate performance of 4.4 Gips. The inter-communication is provided by a lattice of 32 bit buses, where each bus has a 2.5 Mbyte/s bandwidth at 33 MHz clock speed. Thus, the aggregate bandwidth for the whole machine is 2.7 Gbyte/s. The SM's system may utilise a commercially available parallel high performance disk unit with a large capacity. This subsystem could have scalable storage capacity of 15 Gbytes, with access/transfer rate of 36 Mbyte/s. The transferred data is mapped to the array of PEs via a switching network with a high bandwidth.

The computational model of the SM's knowledge representation language consists of two main components: the knowledge base, and the control mechanism. At the abstract level, the knowledge base describes the state of the world, where the controller in conjunction with the PEs and buses, by performing parallel propagation and inheritance, performs the required computation. At a more practical level, the knowledge base (or more precisely, the appropriate portion of the knowledge base) is distributed amongst the rectangular array of PEs, where each frame is mapped onto a

---

[1] A new simulation of the Sheffield Machine ("SM"), was developed using ExperLisp on Macintosh Plus, and the memory had to be extended to cope with the memory-hungry computation involved in the simulation. Further problems were encountered with the under-developed Lisp dialect, which was the only one available at the time (Ritz 1987).

single PE. The control mechanism is also distributed and replicated. That is, every PE, after receiving a data packet, is capable of manipulating its contents (a frame). This manipulation is performed in conjunction with the information brought in by a packet.

The knowledge base consists of a large number of heavily interlinked domains, where each domain is a combination of several tangled hierarchies that are taxonomically structured. At the higher level in each hierarchy, the generic objects contain the general characteristics which can be inherited by individual objects at the lower level. The network is assumed to enjoy all the characteristics that are available to associative nets. The IS-A type link is adopted to represent local and global relationships amongst frames across the entire network. In both global and local relationships, a frame can have more than one child or parent.

The size of the knowledge base is however, too large to be kept in the system's primary distributed memory. Thus, a paging mechanism is employed to retrieve appropriate portions of the knowledge base and to map it to the rectangular array of PEs. Because of the heavily interlinked hierarchies, and the consequent multiparentage, this process may need to be invoked several times after the first retrieval.

In the SM's knowledge base, each node of the network is represented by a frame. Each frame contains a substantial amount of information, which is partly used to define the characteristics of the object that it represents, and partly to determine its relationships with other frames in the network (eg, multiparents/parents-child relationships). In addition, certain information in each frame is used for determining appropriate paths for data packet propagation, and inheritance operations. In contrast to relational database systems frames can represent different type of data and provide exceptions.

Typically, computation in the simulated SM starts with the user creating a query. According to the information given in the query, the appropriate hierarchies are retrieved from the knowledge base and mapped onto the machine's array of PEs.

The propagation starts with the controller sending the relevant information to the appropriate PEs that contain the root node of each mapped hierarchy. Each of these PEs will then create a data packet containing the query and other relevant information, which is then passed to the next level down in the hierarchy. While each packet is being passed down from one PE to another PE, the inheritance operation, if required, will be performed, and its result will be stored in both the packet and the PE. The parallel propagation of data packets and inheritance operations continue until the relevant conditions associated with the type of the given query are met.

Although the propagation of messages is not synchronised, their traversal can be divided into a number of cycles, where each cycle corresponds to a level in the mapped hierarchies. In contrast to serial machines, the process of exploring a large volume of data can be reduced to only a few cycles, which is exactly the same number as the maximum number of levels in a given domain. The analysis of the test runs show that this method of propagation utilises the potential parallelism in the system.

Knowledge processing in the SM involves pattern matching, where frame-names, slot-names and slot-values can have their own respective values, or in their absence, variables to represent them. In the same process, as a result of inheritance operations, variables are bound to appropriate values. In such an environment, two main approaches are adopted for querying the SM's frame-based language; frame-related conjunctive queries, and domain-related conjunctive queries.

In the first method, queries are made with various conjunctive components of a particular frame, where the smallest unit is a slot. Most of the conjunctive clauses in this type of query are properties that are inherited from other hierarchies and domains. In the SM, there are 3 major options available for this type of query.

With the first option, the user can ask for the full definition of an object. Here, only the object name is given and the system will return a fully defined frame which has undergone all the necessary operations including inheritance and pattern matching operations. With the second option, the user can ask for a subset of slots/values of a frame with the frame-name given. With the third option, the user can ask for the name of a frame by providing its full definition, or a subset of its slots/values. It is also possible for the user to provide only a partially defined subset of slots and slot-values. In both cases, the user does not provide the frame-name.

In the second method, domain-related conjunctive queries, a variety of queries can be made. A query can be made related to subclass relationships amongst frames in one or more hierarchies. The user can ask for the relationships between a given frame and its ancestors/descendants. A subtype of this query type involves making a complex query, where two or more slot-names, slot-values (or variables in their absence), are shared in a conjunctive manner between a group of frames.

This type of query enables the user to ask for a wide range of information about the relationships amongst frames in hierarchies, in domains, and in the knowledge base as a whole. The user, for example, can ask for all the frames that share one or more slots. In the absence of known frame-names, slot-names, or slot-values, variables may be given.

In chapter 4 and appendix E, it was demonstrated that in the SM relational operators such as select and project can easily be implemented. It was also demonstrated that it is possible to implement join, but that this would entail a certain increase in time penalties.

Knowledge processing in this simulation is focused on the inheritance of properties and matching operations. The communication is based on message passing which is provided by data packets.

The underlying computational model of the SM is based on the cyclic downward transitive closure of messages (data packets). In this type of propagation, as mentioned above, knowledge processing involves the inheritance and pattern matching operations. The process of passing packets from one PE to its neighbours in the propagation provides communication amongst PEs. Thus, both knowledge processing and communication are intermingled, and constitute the overall operation of the simulation. Furthermore, knowledge processing and communication provide the core elements on which the analysis and evaluation of the simulation are based.

In the course of examining the SM system, a series of tests have been carried out by querying the system. In each query, the time penalties for both knowledge processing and communication are calculated and stored in a file. Each file contains a set of numerical data representing the time penalties on processing relevant frames and their communications. A graphics package has been developed to transform this data into a set of graphs that are used for the analysis and evaluation of the system. Some of these graphs are shown in appendix D.

In the analysis of the test runs, in chapter 6, it was shown that the number of frames that can be serially processed per second is between 160K to 3.1M. In contrast, the number of frames that the SM machine can process in parallel is 6M to 101M frames per second. Although this is certainly a promising result, the figures given for both parallel and serial frame-processing represent two extremes of a spectrum. At one extreme, the time penalty for frame-processing is at its lowest possible level, whilst the other extreme shows the highest level. The end of the spectrum with the lowest time penalties is reached when processing frames with a much smaller number of slots, slot-values, and consequently, with less inheritance and communication. At this extreme, most of the pattern matching operations have failed at the very beginning of their activities thus, less time is spent on pattern matching, inheritance operations and communication. The top end of the spectrum is reached when processing frames with a large number of slots, slot values, and subsequently substantial inheritance operations, and high communication overheads.

By including the time penalty for disk access/transfer, a more realistic result has been produced. The result demonstrates the bottle-neck characteristics of the disk access/transfer operations, which reduced the SM's performance at run-time execution by an overall average of 10 fold. Nevertheless, it can be asserted that the same time penalty applies to serial machines, where the knowledge base is stored on secondary media. However, the figures and charts presented in previous chapters show a series of important points. Firstly, it is obvious that there is a gain in speed-up and, ideally, the target of run-time execution should be that of 101M frames/S.

To achieve the minimum time penalties for execution, it is absolutely necessary to reduce the amount of communication. This is shown clearly in every test-run in appendix D and every related graph shown in chapter 6. One of the most effective ways of reducing the communication overhead is to reduce the number of inheritance operations. This can be achieved by performing inheritance operations at assert time (an analogy for this in neural computing is the substantial amount of time spent at the learning stage). Another way of reducing the time spent on execution is to avoid unnecessary matching. This has already been implemented by the preliminary examination of the contents of the packet and the receiving frame, to see if the inheritance operation is needed, so that further matching and consequent inheritance operations can be avoided.

The method of propagation is another important feature which can play a vital role in achieving high performance. As discussed in chapters 4 and 5, the underlying computational model of the SM has played an extremely important role in achieving high performance and exploiting potential parallelism.

Finally, the implementation of frame-based networks as the knowledge representation formalism has also played an important role in achieving a high performance for the SM. This is, firstly, because of the SM's distributed parallel architecture, which is well suited to the distributed knowledge in a frame-based system. Secondly, a frame-based network offers associativity that provides propagation paths, locality, inheritance of properties, and procedural attachments. Thirdly, each frame in the network can represent a complex concept with all its internal/external relationships, which can be used for identification, retrieval and mapping operations.

7.2 FUTURE WORK

There are several areas in which further research work could be done.

One is the area of identification ie, what mechanism should be employed to identify the relevant frames to a given query. This involves the identification of appropriate portions of the knowledge base which contains those frames. As all the frames in the knowledge base are linked together according to their relationships, as long as an appropriate portion is found the task of finding a particular frame in that portion is trivial (in a parallel environment). In order for the system to identify appropriate portions of the knowledge base, in contrast to data retrieval in database systems, a more flexible approach would be the development of an intelligent front end. Such a system could, by interacting with the user, determine the positions of relevant portions of the knowledge base, and then retrieve them. After the identification and retrieval, the relevant portions will be mapped to the network of PEs on a one-to-one basis. However, the mapping algorithm in the SM can be enhanced with further exploitation of the properties offered by it's frame-based language and other techniques used elsewhere (Bokhari 1981, Moldovan 1986).

The SM's knowledge representation language may be extended and further developed so that it would match sophisticated hybrid languages such as KEE or ART. The extension may include: different parent-child links to represent 'temporal ordering', 'world and belief' systems and 'PARTS-OF' links as well as 'IS-A' links. The query language developed for the SM can also be extended to include the use of universal quantifiers, and functions to define 'max-of', 'min-of', 'sum-of', 'average-of' and 'ordered-set-of' operators. In this enhanced query language the user should be able to make complex queries with unlimited conjoined/disjoined references. The user should also be able to browse through the knowledge base, in parallel. Further development of procedural attachments may be necessary for a more sophisticated response to a given query and for result integration.

The central aim of modifying the hardware of the SM, eg moving the controller out of the rectangular array of PEs, and adding a lattice of buses to the array as described in chapter 3, is to reduce the communication overheads. But care must be taken to avoid any possible contentions within the bus lines. The possibility of more than one PE requiring to use the same route may cause a bottle neck, which would inevitably result in degrading the overall speed of execution. The rich connections provided in the SM may be exploited by a reliable and efficient algorithm for the configuration and inter-connection of the network. This algorithm, in conjunction with the mapping operation, must be fast and efficient. That is, after the controller has identified and retrieved all

the relevant fragments of the knowledge base, and just before the mapping operation starts, the algorithm should have configured an appropriate network interconnection.

Although the test results of the SM's simulation justify optimism, they also show that communication overheads still play an important role in reducing the speed-up of the whole system. In further research work, in pursuit of a more suitable architectural design, attempts were made to study the feasibility of developing/modifying the SM's frame based language for a network of Transputers (Addy 1990, Kennedy 1990, Arran 1991). Transputers offer substantial amount of processing power with full utilisation of local concurrency. In each Transputer, a cluster of closely related frames can be processed concurrently with internal channels representing their relationships, and at the network level; physical serial links are used to provide global relationships with frame-clusters in other Transputers in the network.

The encouraging result of this preliminary study, initiated a new research work in which the SM's frame-based language is currently undergoing further modifications for the adaptation and the utilisation of local concurrency offered in each Transputer, and the exploitation of global parallelism offered by the network (Arran 1992). A certain amount of work has already been done in the development of an appropriate mapping algorithm for mapping a cluster of related frames to each Transputer in the network. An algorithm has also been developed for the propagation of messages which has some similarity to that of the SM's. In the new communication system, attempts have been made to develop an appropriate algorithm for routing the messages amongst the Transputers in the network. This can be further enhanced by the new generation of Transputers: T9000, which has its own built-in routing algorithm at hardware level. The result of this new research work could be a parallel environment where AI applications can be developed, executed and modified at high speed; something that today's serial machines have failed to deliver.

However, in building and developing array/tree based parallel machines, or, broadly speaking, any parallel machine, there are several important issues to be considered. First, in order to optimise the potential parallelism, the amount of serial operations should be reduced to a minimum. Thus, unnecessary shared information should be avoided even at the cost of duplication. Further, the connections between PEs should be studied. Ideally, each PE should be connected to every other PE in the array, so that the amount of communication overheads, in particular contention, will be reduced to a minimum. But, in the absence of appropriate advances in technology, this can be implemented only within limited constraints. As a result, there would be a trade-off between the number of PEs available in the system, the complexity of each PE (ie how much communication can it do while processing other information ?), and the amount

of physical communication lines (eg, buses). One of the consequences of this trade-off is that the number of connections and the number of PEs will be dictated by their relationships. In other words, there would be a limit on the amount of load on a particular bus, and also a limit on how many connection lines a PE can afford to have, so that it can use them effectively. In an efficient system, these relationships must be taken into consideration, which will result in a reduction of the size of the machine to one of manageable size. Next, the issue of global and local control of the system should be considered. That is, the control can be fully (or partially) distributed within the system, or a separate controller can reside outside the array of PEs and dictate its commands to the relevant PEs. The operational dependency of each PE is another issue to be considered here. That is, each PE should be able to independently compute and communicate when necessary.

References & Bibliography

A Datamation Staff Report: " Pushing the State of the Art"
Datamation, Oct. 1985.

Addis T. R. : "Designing Knowledge Based Systems"
Kogan Page Ltd., 1985.

Addy S. V. : " The study and implementation of a Transputer machine to support a distributed frame-based knowledge representation language"
MSc. Dissertation,
School of Computing and Management Sciences,
Sheffield Hallam University, 1990.

Aggarwal A. : "Optimal bounds for finding the max on array of processors with K global buses"
IEEE Trans. on Comp., Vol C-35, No. 1, pp 62-65, Jan 1986.

Albus J. S. : "Brains, Behaviour, and Robotics"
Byte Publication Inc., 1981.

Aleksander I., Morton H. : "An introduction to Neural Computing"
Chapman and Hall, 1990.

Allen J. F. : "Man Machine Dialogues"
Tutorail, 10th Int. Joint Conf. on AI ,
Milano, Aug. 23-28, 1987.

Alty J. L. & Coombs M. : "Expert Systems : Concepts and Examples"
NCC Publications, Manchester, 1984.

Alty J. : "Expert Systems : Techniques and Applications"
Advanced Video Learning,
University of Strathclyde, 1988.

Alvey IKBS Research Theme : "Intelligent Front End"
Work Shop no. 1 and 2, 1984.

Anderson J. R. : "Arguments concerning representations for mental imagery"
Psychological Review 85, 1978.
Andriole S. J. (Ed) : "Applications in Artificial Intelligence"
Petrocelli Books Inc, USA, 1985.

Appalaraju R. : "Parallel Processing Meets Computationally Intensive requirements"
Digital Design, USA, Vol. 14, No. 4, Apr. 1984.

Arran J. : "Investigation into the concurrent execution of a frame-based network"
MSc. Dissertation,
School of Computing and Management Sciences,
Sheffield Hallam University, 1991.

Arran J., Saeedi M. H. : "Parallel execution of Frame-based KBS on Transputers"
Microsystem' 92, 17-19 Nov. 1992, Bratislava, Czechoslovakia.

Attardi G., Simi M. : "Semantic on Inheritance and Attribution in the Description System OMEGA"
AI Memo, No. 642, MIT Lab. 1982.

Bains S. : "Making light work of a 3D optical illusion"
Computing, Oct, 25, 1990.

Bane R. et. al. : " Operating System strategy on ZMOB"
Procs. of IEEE Comp. Society Workshop on Comp. Arch. for Pattern Analysis and Image Data-base Management,
Hot Springs Va. USA, Nov. 11-13, 1981.

Barr A. , Feigenbaum E. A. : "The Handbook of Artificial Intelligence"
vol. 1 and 2,
William Kaufmann Inc., 1981.

Beal R., Jackson T. : "Neural Computing : an introduction"
Adam Hilger, 1990.

Benoit C. , Caseau Y. , Pherivong C. : "Knowledge representation and communication mechanisms in LORE"
Laboratories de Marcoussis Centre de Recherches de la CEG
Route de Nozay 01460, Marcoussis, 1986.

Bertsekas D. P., Tsitsiklis J. N. : "Parallel and Distributed Computers : Numerical Methods"
Prentice-Hall International Editions, 1989.

Beynon-Davise P. : "Expert Database Systems : a Gentle Introduction"
McGraw-Hill (UK), 1991.

Bic L. : " Processing of Semantic Nets on Data-Flow Architecture"
Dept. of Computer Science, University of
California, Irvin, CA92717, USA, 1984.

Black W. J. : "Intelligent Knowledge Base Systems"
Dept. of Computation, UMIST,
Van Nostrand Reinhold (UK) CO. Ltd, 1986.

Bobrow D. G., Stefik M. : "The LOOPS Manual"
Xerox Corporation, 1983.

Bobrow D. G. , Winograd T. : "An Overview of KRL, a Knowledge Representation Language"
in : "Reading in Knowledge Representation"
Edts. : R. J. Brachman, H. J. Levesque,
Morgan Kaufmann Publishers, Inc., 1985.

Bokhari S. H. : "On the Mapping Problem"
IEEE Transaction on Computers, Vol. c-30, no.3, March 1981.

Boyle B. J. : "Novon (a distributed resource network)"
Proc. of Int. Conf. on Cybernetics and Society, IEEE 82 pp 501-505, 1983.

Brachman R. J. : "A structured Paradigm for Representing Knowledge"
PhD thesis,
Harvard University, 1977a.

Brachman R. J. : "What's in a concept : Structural foundations for semantic networks"
International Journal of Man-Machine Studies, 1977b.

Brachman R. J. : " On the Epistemological status of the Semantic Networks"
Bolt Branek and Newman Inc, rep.3807, 1978a.

Brachman R. J. et. al. : "KL-ONE Reference Manual"
BBN Rep. No. 3848, July 1978b.

Brachman R. J., Fikes R. E., Levesque H. J. : "KRYPTON : a functional approach to knowledge representation "
IEEE Computer 16(10), 1983a.

Brachman R. J. : "What IS-A is and isn't: An Analysis of Taxonomic Links in Semantic Networks"
IEEE Computer 16(10), 1983b.

Brachman R. J. , Levesque H. J. : "The Tractability of Subsumption in Framed-Based Description Languages"
Fairchild Laboratory for AI research,
Miranda Avenue, Palo Alto, California 94304, 1983c.

Brachman R. J. : "I Lied About the Trees, Or, Defaults and Definitions in Knowledge Representation"
AI Magazine 6(3), 1985a.

Brachman R. J. , Levesque H. J. : "Reading in Knowledge Representation"
Morgan Kaufmann Publishers, Inc., 1985b.

Bramer M. : "Expert Systems"
7th European Conf. on AI, Tutorial on Expert Systems 21 July 1986.

Bratko I. : "Prolog : Programming for Artificial Intelligence"
Addison-Wesley, UK, 1990.

Brown J. C. : "Simulations of a data flow parallel processor consisting of a rectangular array of processor nodes"
Contribution to Esprit 1984 workplan, in distribution to Esprit contributors, Oct. 1983.

Brown J. C. , Saeedi M. H. : "A parallel Multiprocessor for Knowledge bases"

School of Computing and Management Sciences,
Sheffield Hallam University, 1986.

Brown J. C. : "Taxonomic Hierarchies with Attribute Inheritance : Potential and realisable Parallelism"
Workshop on Parallel Architectures, BCS Parallel Processing and Occam User Groups, Birkbeck College London, 1988.

Brown J. C., Saeedi M. H. : "A Wafer-Scale Integration Architecture for Frame Networks"
SERC/IED Parallel and Novel Architectures Club,
Workshop of Handling Large Knowledge Bases Declaratively, Sept. 1989.

Brownston L. et al, : "Programming Expert Systems in OPS5"
Addison-Weslsy, USA, 1985.

Carbonnell J. C. , Langley P. : "Learning and knowledge acquisition"
7th European Conf. on AI, Tutorial, 21 Jul. 1986.

Charniak E., McDermott D. V. : "Introduction to Artificial Intelligence"
Addison-Wesley, USA, 1985.

Charniak E., Riesbeck C. K., McDermott D. V., Meehan J. R. : "Artificial Intelligence Programming"
Lawrence Erlbaum Associates, USA, 1987.

Charniak E. : " Passing Markers : A theory of Contextual Influence in Language Comprehension"
Cognitive Science 7, 171-190, 1980.

Chomsky N. : "Syntactic Structures"
Mouton : The Hague, 1957.

Chung P. W. H., Kingston J. K. C. : "State of the art - Knowledge-based system toolkits"
AIAI, August 1988.

Clancey W. J. and Shortliffe E. H. : "Readings in Medical Artificial Intelligence : the first decade"
Reading, Ma, Addison-Wesley, 1984.

Classe A. : "Speed limits"
Computing, Aug. 9, 1990.

Clayton B. D. : "Inference ART : Programming Tutorial"
Vol. 1, Elementary ART Programming,
Ferranti Computer Systems, 1987.

Clocksin W. F., Mellish C. S. : "Programming in Prolog"
Springer-Verlag, West Germany, 1984.

Codd E. F. : "The Relational Model for Database Management"
Version 2, Addison-Wesley, 1990.

Cohen P. R. , Feigenbaum E. A. : "The Handbook of Artificial Intelligence"
William Kaufmann Inc., 1981.

Cohn G., Hayes P. : "Qualitative reasoning"
7th European Conf. on AI, Tutorial, 21 Jul, 1986.

Collins A. M. , Loftus E. F. : " A spreading Activation theory of Semantic Processing"
Psychological review, Vol. 82, No. 6, 1975.

Darlington J. , Reeve M. : " Alice and the Parallel Evaluation of Logic Programs"
10th Int. Symp. on Comp. Arch., 1983.

Date C. J. : "An Introduction to Database Systems"
Vol. I,
Addison-Wesley, USA, 1984.

Date C. J. : "An Introduction to Database Systems"
Vol. II,
Addison-Wesley, USA, 1986.

Davis A. L. , Robinson S. V. : "the Architecture of the FAME-1 Symbolic
Multiprocessing System"
Procs. of Int. Joint Conf. on Arch., 1985.

Davis R. , King J. : "An overview of production systems"

Computer Science Department, Stanford University, 1980.

Davis R., Lenat D. B. : "Knowledge-based Systems in Artificial Intelligence"
McGraw-Hill, USA, 1982.

Deen S. M. : "Principles and Practice of Database Systems"
Macmillan Publishers Ltd., UK, 1985.

Decegama A. L. : "The Technology of Parallel Processing"
Prentice-Hall International Editions, 1989.

Delgado-Frias J. D. , More W. R. : "multiprocessor architectures for Artificial Intelligence"
Report no. OUEL 1671/87.
University of Oxford, Dept. of Engineering Science,
VLSI Design Research Group, Parks Road,
Oxford, 1987a.

Delgado-Frias J. D. , More W. R. : "Semantic network Processing Systems"
Report no. OUEL 1677/87.
University of Oxford, Dept. of Engineering Science,
VLSI Design Research Group, Parks Road,
Oxford, 1987b.

Deitel H. M. : "Operating Systems"
Addison-Wesley Publishing Company, 1990.

Dixit V. , Moldovan D. I. : " Discrete Relaxation on SNAP"
Proc. 1st conf. on AI Applications,
Cat No. 84CH2107-1,
Denver co USA, Dec. 1984.

Domesday Project,
MICRO USER,
Jan., pp32-37, 1987.

Donovan J. J. : "System Programming"
McGraw-Hill, Japan, 1983.

Duda R. O., Reboh R. : "AI & Decision-Making : The PROSPECTOR Experience"
In :"Artificial Intelligence Applications for Business"
Reitman W.,
Norwood N. J., Ablex, 1984.

Durham T. : "Moving Experts Forward on Small Steps at a Time"
Computing, Sept. 1987.

Durham T. : "The Parallel Path to a Thinking Machine"
Computing, April 23, 1987.

Erman L. D. : "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty"
ACM Computing Surveys, Vol. 12, 1980.

Ritz D. : "ExperLisp"
ExperTelligence, inc., 1987.

Fahlman S. E. : "NETL: a system for representing and using real-world knowledge"
MIT Press, 1979.

Fahlman S. E. : "Parallel Processing in Artificial Intelligence"
Computer Science Dept.,
Carnegie-Mellon University,
Pittsburgh, PA 15213, USA, 1981.

Fahlman S. C. , Touretzky D. S. , Roggen W. van : "Cancellation in Parallel Semantic Network"
In : "Parallel Computing : Special Issue on Parallel Processing"
Edts. : G. S. Almasi, G. Paul, North-Holland-Amsterdam, 1985a.

Fahlman S. E. , Hinton G. : "Massively Parallel Architectures for AI : NETL, THISTLE, and Boltzmann Machines"
Computer Science Dept.,
Carnegie-Mellon University,
Pittsburgh, PA 15213, USA, 1985b.

Fahlman S. E. , Hinton G. E. : " Connectionist Architectures for Artificial Intelligence"
Computer, Jan. 1987.

Feldman J. A. : " Connections"
Byte, Vol. 10, No. 4, April 1985.

Filmore C. J. : "Toward a modern theory of case"
In : "Modern Study in English : Reading in Transformational Grammar"
Edts : D. A. Reible, S. A. Schane,
Prentice-Hall, 1966.

Findler N. V. : "Associative Networks : Representation and Use of Knowledge by Computers"
Academic Press, 1979.

Fogelman-Soulie F. : "Artificial neural networks"
Computing Beyond 2000, THE SIXth GENERATION, Nov., 1990.

Foremski T. : "The future displayed in a crystal"
Computing, Oct. 18, 1990.

Forgy C. L. : "Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem"
Artificial Intelligence, Vol. 19, no. 1, Sept 1982.

Fox B. : " Optical Storage Come to Order"
New Scientist, Nov. 27, 1986.

Fox M. S. : " On Inheritance In Knowledge representation"
Dept. of Computer Science,
Carnegie-Mellon University,
Pittsburgh, PA 15213, USA, 1986.

Fox G. et al. : "Soving Problems on Concurrent Processors"
Prentice-Hall, 1988.

Freeman R. : "Classification of the Animal Kingdom"
Hodder and Stoughton Ltd.,
The Reader Digest"s Association Ltd., 1985.

Fuchi K. : "The Direction the FGCS project will take"

New Generation Computer (Japan)
Vol. 11, No. 1 p 3-9, 1983.

Ffuruhawa K. et. al. : "Problem Solving and Inference Mechanism"
Procs. of 10th Int. Conf. on 5th Generation, Oct. 19-22, 1981.

Forsyth R. (ed.) : "Expert systems : principles and case studies"
Chapman and Hall, 1985.

Frost R. A. : "Introduction To Knowledge Base Systems"
Collins, GB, 1986.

Gaschnig J. :"PROSPECTOR : An expert systems for mineral exploration"
In : "Reading in Expert Systems",
Gordon & Breach, New York, 1982.

Goldberg A. : "Smalltalk : The Interactive Programming Environment"
Addison-Wesley, USA, 1984.

Graham N. : "Artificial Intelligence"
Tab Books Inc., USA, 1979.

Granger C. : "Fuzzy reasoning in knowledge based system for object classification"
Inria, Centre de Recquencourt le Chesnay Cedex, France, 1986.

Gray E. M. D. : " Comparison of Hardware Assistance to database and IKBS"
Proceedings of the Second Workshop for the Special Group on Knowledge Manipulation Engines (SIGKME),
May 1987.

Gray J. P. : "The choice of a suitable benchmark knowledge base for knowledge representations based on taxonomic classification with attribute inheritance"
MSc. Dissertation,
School of Computing and Management Sciences,
Sheffield Hallam University, 1987.

Green J. : "An Extended Dynamic Dataflow Multiprocessor"
B.Sc. Project Report,

School of Computing and Management Sciences,
Sheffield Hallam University, 1984.

Gupta A. : "Parallelism in Production Systems"
Proces. 5th Int. Workshop Expert System & Their Applications,
Avignon France, Vol. 1, pp25-57, 1985.

Hall P. J. : "How to Solve it in Lisp"
Sigma Press, UK, 1989.

Harrison P., Reeve M. : "Parallel Architectures for Non-Sequential Programming"
Int. Report,
Dept. of Computing,
Imperial College, Univ. of London, 1986.

Hayes F. : "The Crossbar Connection"
BYTE, Nov. 1988.

Hayes J. E., Michie D. : "Intelligence Systems : The Unprecedented Opportunity"
Ellis Howood Ltd., UK, 1984.

Hays-Roth F., et al : "Building Expert Systems"
Vol. I, Addison-Wesley, USA, 1983.

Hayward E. : "High on speed"
Computing, March 14, 1991.

Hekmatpour S. : "Introduction to Lisp and Symbol Manipulation"
Prentice Hall, UK,1988.

Hendrix G. G. : "Expanding the utility of semantic networks through partitioning"
Proc. 4th IJCAE, Morgan Kaufmann, 1975.

Hennessy J. L., Patterson D. A. : "Computer Architecture a Quantitative Approach"
Morgan Kaufmann Pub., Inc., 1990.

Hewitt C. E. : "The apiary network architecture for knowledgeable systems"
Proc. of Lisp Conf., Stanford, pp 107-117, 1980.

Hewitt C. E., Attardi G., Simi M. : " Knowledge embedding in the Description System OMEGA"
Procs. of 1980 AAAI Conf., Stanford 1980.

Hillis D. , Steele G. : "Data Parallel Algorithms"
Communication of the ACM,
Vol. 29, No. 12, Dec. 1986.

Hillis W. D. : "Connection machine"
MIT Press, 1985.

Hinton G. E., Sejnowski T. J., Ackley D. H. : "A learning Algorithm for Boltzmann Machines"
Tech. Report, CMU-CS, May 1984.

Hird B. : "Modification to a data flow multiprocessor for AI Applications"
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1985.

Hollingum A. : "A Supervisory System for a Dataflow Multiprocessor"
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1985.

Hwang K. , Ghosh J. , Chowkwanyun R. : "Computer Architectures for Artificial Intelligence"
Computer, Vol. 20, No. 1, Jan. 1987.

Hwang K., Briggs F. A. : "Computer Architecture and Parallel Processing".
McGraw-Hill Book Company, 1987.

Hyman . A, : "Charles Babbage - computer pioneer"
The Computer Bulletin, vol. 3, June 1991.

Ibbett R. N., Topham N. P. : "Architecture of High Performance Computers"
Macmillan Computer Sciences Series, 1989.

Intel Scientific Computers : "Intel iPSc : Sale Literature"

Greenbrier Parkway,
Beaverton, Oregon 97006, 1986.

IntelliCorp : "KEE^{TM} : Software Development System User' s Manual"
KEE Version 3.0, July25, 1986.

Jelly I. : "A Parallel Model and Architecture for a Pure Logic Language"
PhD thesis,
School of Computing and Management Sciences,
Sheffield Hallam University, Oct. 1990.

Jackson P. : "Introduction to Expert Systems"
2nd edition,
Addison-Weslsy, UK, 1990.

Jones J. , Millington M. , Ross P. : "A Blackboard shell in Prolog"
Dept. of AI, University of Edinburgh.

Jonson L., Keravnou E. T. : "Expert Systems Architectures"
Kogan Page Ltd., UK, 1988.

Kaczmarek T. S. : "Recent developments in NIKL"
Proc. AAAI-86,
Morgan Kaufmann, 1986.

Kanal L., Kumar V. : "Search in Artificial Intelligence"
SpriNger-verlag, USA, 1988.

Kennedy H. : "The design and implementation of an Application Development Tool for a parallel Transputer-based machine"
MSc. Dissertation,
School of Computing and Management Sciences,
Sheffield Hallam University, 1990.

Kolodner J. L., Riesbeck C. K. : "Experience, Memory and Reasoning"
lawrence Erlbaum Associates, USA, 1986.

Krishnamurthy E. V. : "Parallel Processing : Principles and Practice"
Addison-Wesley, 1989.

Laurent J. , Ayel J. , Thome F. , Ziebelle D. : "Comparative Evaluation of Three Expert System Tools : KEE, Knowledge Craft, ART"
Labiratorie la Informatique appliquee, University de Savoie,
BP 1104 730011 Chambery, France, 1988.

Lavington S. H. : " a Technical Overview of the Intelligence File Store (FS)"
Internal report, IFS/6/87,
Lavington, Dept. of Computer Science,
University of Essex, Wivenhoe Park,
Colchester CO4 3QS, 1987.

Lehr T. F., Wedig R. G. : "Towards a GaAs Realization of a Production-System Machine "
IEEE Computer 20(4), 1987.

Lenat D. B., Feigenbaum E. D. : " on the Threshold of Knowledge"
Procs. of 10th Int. Joint Conf. on AI,
Milano, Aug. 23-28, 1987.

Loh M. C. , Brown J. C. : "A data flow parallel processor consisting of a rectangular array of processor nodes"
Leeds University Conference on 25 years of Computing, 1982a.

Loh M. C. : "Program relocatability and Multiprogramming in a dataflow computer"
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1982b.

Markov A. A. : "Theory of algorithms"
National Academy of Science, Moscow, USSR, 1954.

McClelland J. L., et al : "Parallel Distributed Processing" Vol. 1,
The MIT press, USA, 1987.

McDermott J. R. : "A rule-based configurer of computer systems"
Artificial Intelligence, Vol 19 (1), 1982.

McDermott J. R. : "Building Expert Systems"
In :"Artificial Intelligence Applications for Business"
Reitman W.,
Norwood N. J., Ablex, 1984.

McFadden F. R., Hoffer J. A. : "Database Management"
Benjamin/Cummings Publishing Company, Inc., USA, 1991.

Meinel C. : "Structural investigations of parallel computing devices"
Computing and AI (Czechoslovakia), vol 3 no 4 , pp331-40, 1984.

Metropolis N., et al (Ed) : "A History of Computing in the Twentieth Century"
Academic Press, UK, 1980.

Michie D. : " Introductory Reading in Expert Systems"
Gorden and Breach, 1984.

Michalski R. S. : " Machine Learning Tutorail"
10th Int. Joint Conf. on AI,
Milano, Aug. 23-28, 1987.

Mill J. : "Doing the knowledge"
Computing, Feb. 15, 1990.

Minker J. , Asper C. : "Parallel problem solving on zmob"
Proc. Conf. of Trends & Applications 1983.
Automating Intelligent Behaviour. Appl. & Frontiers,
Gaithersburg Md. USA, p 142-6, May 1983.

Minsky M. : "A framework for representing knowledge"
MIT, AI Lab, June 1974.

Minsky M. : " K-Lines: a theory of Memory"
Cognitive Science, Vol. 4, 1980.

Moldovan D. I. , Tung Yu-Wen : " SNAP: A VLSI Architecture for Artificial Intelligence Processing"
Journal of parallel and distributed computing 2, 1985.

Moldovan D. I. : "Partitioning and mapping algorithms into fixed size systolic arrays"
IEEE Trans. on Comp. Vol. C-35, No. 1, pp 1-12, Jan 1986.

Moon D. : "Architecture of the Symbolics 3600"
Procs. of 12th Interest Group on Comp. Arch., 1985.

Moore R. , Richie G. : "Natural Language Processing"
7th European Conf. on AI, Tutorial, 21 July 1986.

Morris N. M. : "An Introduction to : 8086/88 Assembly Language Programming for Engineers"
McGraw-Hill Book Company, 1987.

Motta E. , Eisenstadt M. : " KEATS: the Knowledge Engineer's Assistant"
Rep. No. 20,
Human Cognition Research Laboratory,
Milton Keynes MK7 6AA, UK, Dec. 1986.

Nelson R. J. : "Introduction to Automata"
John Wiley & Sons, Inc, USA, 1968.
Newell A., Simon H. A. : "Human Problem Solving"
Prentice-Hall, 1972.

Newell A., Simon H. A. : "Computer Science as Empirical Enquiry"
Communications of the Association for Computing Machinery (19)3, 1976.

Nii P. H. : "Blackboard systems Part II, Blackboard Applications Systems, Blackboard systems from a Knowledge Engineering Perspective"
AI Magazine, August 1986.

Owen S. : "Heuristic for analogy matching"
Dept. of AI University of Edinburgh, EH1 1HN, Scotland, 1987.

Oyama T., Ogawa Y., Sugiyama K. : "F6490 Magnetic Disk Subsystem : DIA"
FUJITSU Scientific Journal, 26, 4, Feb., 1991.

Patterson D. W. : "Introduction to Artificial Intelligence and Expert Systems"
Prentic-Hall, 1990.

Pfaltz J. L. : "Computer Data Structures"
McGraw-Hill Book Company, 1977.

Post E. LL. : "Formal reductions of the general combinatorial decision problem"
American J. Mathematics 65, 1943.

Pradhan D. K. : "Correction to fault-tolerant multiprocessor link and architecture"
IEEE Trans., Vol. C-35, No. 1, Jan 1986.

Pollack B. W. (Ed) : "Compiler Techniques"
Auerbach Publishers, USA, 1972.

Potter J. L. : "The Massively Parallel Processor"
Research Report,
MIT, 1980.

Pratt V. : "Thinking Machines : The Evolution of Artificial Intelligence"
Basil Blackwell Ltd., UK, 1987.

Quillian M. R. :" Semantic Memory"
In : "Semantic Information Processing"
Edt. : M. Minsky, MIT Press, 1968.

Reeve M., Zenith S. E. : "Parallel Processing and Artificial Intelligence"
Willey, 1989.

Reissman I. : "Graphics Extension to a Simulation of a Dataflow Multiprocessor"
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1987.

Reilly R. J. : "A Connectionist Model..."
Procs. of 10th Int. Conf. on Comp. Linguistics, Stanford Ca. USA,
July 2-6, 1984.

Rich E. : "Artificial Intelligence"
McGraw-Hill, 1983.

Ringland G. A., Duce D. A. : "Approaches to Knowledge Representation : An Introduction"
John Wiley & Sons Inc., 1989.

Roberts R. B., Goldsmith I. P. "The FRL Primer"
AI Memo, no. 408, MIT Lab, Cambridge, USA, 1977.

Roche S. : "Moving closer to the fifth generation"
Computing, Feb. 8, 1990.

Robbins C. : "Keeping track of all the transistor technology"
Computing, Feb. 22, 1990.

Rumelhart D. E., et al : "Parallel Distributed Processing"
Vol. 2,
The MIT press, USA, 1987.

Sapatty P. S. : " On the Efficiency of Structural Realisation of Operations Over Semantic Network"
Engineering Cybernetic, (USA),
Vol. 21, No. 5, Sept.- Oct. 1983.

Schank R. C. : "Conceptual Dependency : A theory of natural language understanding"
Cognitive Psychology 3, 1972.

Schank R. C., Abelson R. P. : "Script, Plans, Goals and Understanding"
Lawrence Erlbaum Associates, USA, 1977.

Schank R. C. : "Dynamic Memory"
Cambridge University Press, 1982.

Schubert L. K. : " Extending the Expressive Power of Semantic Networks"
Artificial Intelligence 7, 163-198, 1976.

Schutzer D. : "Artificial Intelligence : An Applications-oriented Approach"
Van Nostrand Reinhold Company, USA, 1987.

Searle J. R. : "Minds, Brains and Science"

The 1984 Reith Lectures, British Broadcasting Corporation, 1984.

Seitz C. L. : "The Cosmic Cube"
Communications of the ACM, vol. 28, no. 1, Jan. 1985.

Sell P. S. : "Expert Systems: a practical introduction"
Macmillan Ltd. 1983.

Shields M. W. : "An Introduction to Automata Theory"
Blackwell Scientific Publications, GB, 1987.

Shortliffe E. H. : "Computer-Based Medical Consultation : MYCIN"
Newyork, Elsevier, 1976.

Simon H. A., Newell A. : "Computer Augmentation of Human reasoning"
Spartan Books, 1965.

Socrates : "a Toolkit for Expert systems"
GEC Reseach Ltd., 1987.

Stanfill C. , Kahle B. : "Parallel Free-Text Connection Machine System"
Communication of the ACM,
Vol. 29, No. 12, Dec. 1986.
Stolfo S. J. : " On the Design of Parallel Production Systems Machines"
Procs. of 18th Hawaii Int. Conf. on System Sciences, 2-4 Jan 1985.

Stolfo S. J. : "Initial Performance of the DADO2 Prototype"
Computer, Vol. 20, No. 1, Jan 1987.

Tabak D. : "Multiprocessors"
Prentice-Hall, 1990.

Treleaven P. C. , Refenes A. N. , Lees K. J. : " Computer Architecture for Artificial Intelligence"
Technical Rep. no. 119,
Dept. of Comp. Science,
University of College London, 1986.

Touretzky D. S. : "The Mathematics of Inheritance Systems"

Pitman, UK, 1986.

Uhr L. : "Psychological motivation and underlying concepts"
in : "Structured Computer Vision"
Edts. : S. Tanimoto, A. Klinger,
Academic Press, 1980.

Walder B. : "Intelligent Services"
Personal Computer Magazine, Oct. 1992.

Walker D. : "Dynamic Simulation of Dataflow Multiprocessor "
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1983.

Walters J. R., Nielsen N. R. : "Crafting Knowledge based Systems ..."
John Wiley & Sons, USA, 1988.

Waterman D. A. : "A guide to Expert Systems"
Addison-Wesley, Reading, Ma, 1985.

Weiss R. : "Computer Architecture : varied blueprints will lift system speed to dizzying heights"
Electronic Design (USA) Vol. 33,
No. 12 pp82-92, 1985.

Wellsch K. , Jones M. : "Computational analogy"
Logic programming and AI group
University of Waterloo,
Waterloo, Ontario, N2L 3G1, Canada. 1984.

White I. N. : "Extensions to Supervisory System for a Dataflow Multiprocessor"
B.Sc. Project Report,
School of Computing and Management Sciences,
Sheffield Hallam University, 1986.

Whyte L. L. , Wilson A. G. : "Hierarchical structures"
American Elsevier publishing Company, Inc. , 1983.

Wilensky R. "LISP craft"
W. W. Norton & Company, USA, 1984.

Wilensky R. : "Hodiak : a knowledge representation language"
Procs. of the 6th annual Conf. of the Cognitive Science society,
Bouldor Colorado, 1984.

Winograd T. : "Frame Representations and the Declarative/Procedural Controversy"
in : "Reading in Knowledge Representation"
Edts. : R. J. Brachman, H. J. Levesque,
Morgan Kaufmann Publishers Inc., 1985.

Winston P. H. : "Learning Structured Description from Example"
In "Psychology of Computer Vision"
Edt : P. H. Winston,
McGraw-Hill, 1975.

Winston P. H. : " Artificial Intelligence"
Addison-Wesley, 1977.

Winston P. H., Horn B. K. P. : "Lisp"
Addison-weslsy, USA, 1984.

Woods W. A. : " Research in Language Understanding"
Progress rep. No.2,
Bolt Branek and Newman Inc, rep.3797, 1978.

Woods W. A. : " What's Important About Knowledge Representation ?"
Computer, Oct. 1983.

Woods W. A. : "What is a Link : Foundation for Semantic Network"
In : "Representation and Understanding"
Edts : D. G. Bobrow, A. Collins, 1985.

Yazdani M. : " Artificial Intelligence : Principles and Applications"
Chapman and Hall, 1986.

Zadeh L. A. : "Fussy logic and its applications to approximate reasoning"
Information Processing, North-Holland, 1974.

APPENDIX A : CODING

In this appendix some of the top level functions developed for the simulation are presented.

{**********}
This function will load the system. It initialises all the arrays and global variables. It reads the disk contents into an array called 'disk_array'. It then load the contents of 'disk_index_file' into array 'disk_index'. It loads kb_first and kb_second and after reading the contents of clash_file it will binds them to a global variable 'list_of_clashes.

```
(defun load_system ()
(setq file_names '(
   main-disk:lisp:all_files:creat_disk_file
   main-disk:lisp:all_files:hash_file2
   main-disk:lisp:all_files:creating_frame_file1
   main-disk:lisp:all_files:map_file2
   main-disk:lisp:all_files:menu_file
   main-disk:lisp:all_files:menu_file2
   main-disk:lisp:all_files:propagate_file
   main-disk:lisp:all_files:read_write_file
   main-disk:lisp:all_files:query-domain-1
   main-disk:lisp:all_files:query-domain-2
   main-disk:lisp:all_files:query-domain-3
   main-disk:lisp:all_files:time_delay_file
   main-disk:lisp:all_files:xxx-file
   main-disk:lisp:all_files:query-object-one
   main-disk:lisp:all_files:query-object-two
   main-disk:lisp:all_files:query-object-three
   main-disk:lisp:all_files:utility-file
   main-disk:lisp:graphics-folder:graphics_file0
   main-disk:lisp:graphics-folder:graphics_file01
   main-disk:lisp:graphics-folder:graphics_file1
   main-disk:lisp:graphics-folder:graphics_file
   main-disk:lisp:graphics-folder:Graph-Rect-PEs1))

   (while (not(null file_names))
      (load_file (car file_names))
      (print `(File : ,(car file_names) is loaded.))
      (setq file_names (cdr file_names)))
```

List_of_clashes is a global association list, which would contains the list of clashes at the time of hashing process (if any).

```
(setq list_of_clashes nil)
```

```
(setq s02 (open_read "main-disk:lisp:coordinates"))
(setq coords (read s02))(close s02)
```

coords is a golbal varibale which contains the contents of the file 'coordinates'. This file contains the coupled coordniates of rectangular array of PEs and the Graphics-screen.

```
(setq The_Propagation_Paths nil)
```

This is a global var which contains the paths of propagation at each run. It will be bound to a value (paths of propagation) at each run.

The FREE variable (global variable) Free_maphistory would represents the last (row * column) of each level of an already mapped hierarchy. If its content is nil, it represents that there has been no mapping.

```
(setq  Free_maphistory  0)
```

This array represent the disk .

```
(setq    disk_array  (make-array  '(40  5)))
```

This array represents the controller's copy with addition

```
(setq  controller_copy  (make-array  '(2000  4)))
```

This array represents the disk_index, which constitute the main part of the controller's copy .

```
(setq  disk_index  (make-array  '(2000  4)))
```

This array represents the rectangular array of PEs.

```
(setq  PEs  (make-array  '(5  40)))
```

writing disk_index_file into disk_index array.

```
(write_array_2   disk_index   'main-disk:lisp:kb_files:disk_index_file)
```

loading all the levles of the KB.

```
(load_disk  4  0  4  'main-disk:lisp:kb_files:disk_level4)
loading  level  1.
(load_disk  3  0  12  'main-disk:lisp:kb_files:disk_level3)
loading  levle  2.
(load_disk  2  0  25  'main-disk:lisp:kb_files:disk_level2)
loading  level  3.
(load_disk  1  0  32  'main-disk:lisp:kb_files:disk_level1)
loading  level  4.
(load_disk  0  0  7  'main-disk:lisp:kb_files:level0_1)
loading  lowest  level  of  a1.
(load_disk  0  7  14  'main-disk:lisp:kb_files:level0_2)
loading  lowest  level  of  p1.
(load_disk  0  14  21  'main-disk:lisp:kb_files:level0_3)
loading  lowest  level  of  v1.
(load_disk  0  21  34  'main-disk:lisp:kb_files:level0_4)
loading  lowest  level  of  z1.
```

loading the primary kb.

```
(load-text   'main-disk:lisp:kb_files:kbs_first)
(load-text   'main-disk:lisp:kb_files:kbs_second)
```

reading clash_file.

```
(setq  path2  (open_read  (getprep  'main-disk:lisp:kb_files:clash_file))
       list_of_clashes  (read  path2))
(close  path2)
(print  "The_System  is  now  available")   )
```

{**********}

The following function returns a hierachically structured list of object-names.

```
(defun get_list (root_name)
  (append (list(list root_name))
          (get_propts (list root_name) nil)))
```

{**********}

This function reads information from the primary KB and creates the disk_array, and writes the object-names to it in an specified order.

```
     (defun   creat_disk (text_files root_names)
       (while (not(null text_files))
       (load-text (car text_files)) ;;loading the primary KBs
       (setq text_files (cdr text_files)))
       (do_load root_names 0 0))
```

{**********}

This function performs hashing.

```
(defun hashing (word)
    (cond    ((null word) 'ERROR_IN_HASHING)
             (t (get_hash_no10 (hash10   (explode1 word) 0)))))
```

{**********}

This function is used by the ADT (Application Development Tool) to create a frame.

```
(defun create_frames0 ()
(create_frames1 (read(print '(type in frames name)))))
```

{**********}

This function performs the mapping operation (if required).

```
(defun check_mapping (full_frame) ;the full_frame is the inj_point.
  (cond ((and(equal (cadr(second(third(second(second full_frame)))))
            'none)
            (null(aref controller_copy (caar full_frame) 3)))
            (print `(Frame_ ,(cadar full_frame) is a Root_Node))(terpri)
            (map_to_rect (cadar full_frame))) ;perform mapping.
        ((and(not(equal(cadr(second(third(second(second full_frame)))))
            'none))
            (null(aref controller_copy (caar full_frame) 3)))
            (print `(Frame_ ,(cadar full_frame) is Not_a Root_Node))
            (map_to_rect (cadar full_frame))) ;perform mapping.
        (t (print "No_Mapping is required."))))
```

{**********}

The following functions creates a new menu.

```
(setq query_menu (newmenu 10 "Queries"))
(setq menulist (append '(((10 1) (help))
                         ((10 2) (query_object))
                         ((10 3) (query_domains)))
                       menulist))
(setq create_frames (newmenu 11 "CreateFrame"))
(appendmenu create_frames "creating_frame;add_slots;delete_slots")
(setq create_graphics (newmenu 12 "Graphics"))
(appendmenu create_graphics "do_graphics")
```

{**********}

This function takes the object's name and finds its frame in the knowledge base.

```
(defun find_object (object_name &aux temp1 temp)
(cond  ((null (setq temp1 (check_for_clashes object_name)))
        (cond ((not(setq temp (change_to_hash object_name)))
            (print `(The object ,object_name is not available in the base))
          nil)
              ((null(car temp))(print '(Corrupted data in Disk_Index))nil)
              ((null(cadr temp))(print '(Corrupted data in Disk_Index))nil)
        (t (aref disk_array (car temp) (cadr temp)))))
       (t(aref disk_array (aref disk_index (car(last temp1)) 2)
                    (aref disk_index (car(last temp1)) 3)))))
```

```
{**********}
 This function finds the injection point for data-packet propagation.

(defun get_injection_point (frame)
 (print '(Finding the injection point.))
 (setq number (caaaar (cadr(fourth frame)))
          inj_frame(aref disk_array (aref disk_index number 2)
                                    (aref disk_index number 3))
           result (time_register counter20 (tickcount)))
 (print `(The injectionn point is : ,(cadr(first inj_frame))))

{**********}
This function is the top level function for domain related optin 2.

(defun query_Domain_2 (frame_name &aux query_frame frame)
(setq query_frame (create_aframe frame_name)
          frame (aref disk_array(aref disk_index (caar query_frame) 2)
                                (aref disk_index (caar query_frame) 3)))
  (terpri)(print '(Type_in file_name for your time delay))
  (setq td-file (read))
  (terpri)(print '(Type_in file_name for prompts))
  (setq td-file2 (read))
  (setq p0 (open_append (getprep 'main-disk:lisp:timedelay-
  folder:qandfilename)))
  (print (list query_frame td-file 'Off3)p0)(close p0)
  (Domain_30 frame (cadr query_frame)))

{**********}
This is the top level function for query-object of the first type.

(defun Query_Object_1 (frame_name &aux frame)
(cond((setq frame (find_object frame_name))
      (terpri)
       (print '(Type_in file_name for your time delay))
       (setq td-file (read))
         (setq p0 (open_append (getprep 'main-disk:lisp:timedelay-
     folder:qandfilename1)))
       (print (list td-file 'Q1)p0)(close p0)
;wirtes the query and file name to qandfilename1 file
        (print(list frame_name (third(new_q10 frame)))))
;this returns the query-frame after all the necessary operations
;have been performed on it.
     (t "Try-Again")))

{**********}
This is the top level function for query-object of the second type.

(defun Query_Object_2 (frame_name &aux query_frame frame)
(cond  ((setq frame (find_object frame_name))
        (setq query_frame (create_aframe10 frame))
          (terpri)(print '(Type_in file_name for your time delay))
          (setq td-file (read))
          (setq p0 (open_append (getprep 'main-disk:lisp:timedelay-
          folder:qandfilename2)))
          (print (list query_frame td-file 'Q2)p0)(close p0)
;wirtes the query and file name to qandfilename2 file
        (print(list frame_name (third(new_q20 frame (cadr
        query_frame))))))
;this returns the query-frame after all the necessary operations
```

```
;have been performed on it.
     (t  "Try-Again")))
{**********}
```
This is the top level function for query-object of the thrid type.

```
(defun  Query_Object_3  (&aux  frame)
(cond  ((terpri)
       (Interact)(get_Frame)
       (print '(Type_in file_name for your time delay))
       (print '(Note : all the time-delay files have a prefix of main-
disk:lisp:timedelay-folder:td))
       (print '(eg : main-disk:lisp:timedelay-folder:td1 where td1 is a file name))
       (setq  td-file  (read))
       (print  '(type-in  file-name  for  prompts))
       (setq  td-file2  (read))
       (setq  p0  (open_append  (getprep  'main-disk:lisp:timedelay-
folder:qandfilename3)))
       (print  (list  td-file  'Q3)p0)(close  p0)
;wirtes the query and file name to qandfilename3 file
       (print(list  frame_name  (third(new_q30  frame)))))
;this returns the query-frame after all the necessary operations have been performed on
it.
     (t "Try-Again")))
```

```
{**********}
```
The following top level functions are used for calculating the time delay for inheritance, matching and communication.

```
(defun  do_matchdelay  (list1  list2)
(setq  Free_matchdelay  0)
(do_matchdelay0  list1  list2)  Free_matchdelay)

(defun  transfer_time  (packet-size  distance)
(*  packet-size  0.1  distance))

(defun  packet_size  (packet)
(setq  result_size  0)
(calculate_size  packet)  result_size)
```

```
{**********}
```
This is the high level function for graphics package.

```
(defun  do_graphics  (&aux  file_name  file_name1  (file_contents  nil))
(terpri)(print  '(Type_in  your  time-delay  file_name))
(setq  file_name  (read))
(setq  graph_port  (open_read  (getprep  file_name)))
(print  `(Reading-  ,file_name  ))
(while(not(end_of_file  graph_port))
(setq  file_contents  (append  (list(read  graph_port))file_contents)))
(close  graph_port)(reverse  file_contents)
(print  '(Type_in  time-delay  file_name  without  prefix  :))
(setq  file_name1  (read))
(call_graphs  (reverse  file_contents)  file_name1))
```

```
{**********}
```

APPENDIX B : BENCHMARK KNOWLEDGE BASE

In this appendix, the benchmark knowledge base developed for the simulation is presented. The information in this knowledge base is in abstract form, in an attempt to capture most of the complexities of real world knowledge.

```
(((1749   a1)
  (id_slots ((offspring_no (value 3 ))
          (level_no (value 1))
          (parents (value none))
             (children (value ((681 a11)(781 a12)(881 a13))))))
   (ind_slots (a1_ind_slot1 (value a1_ind_v1))
            (a1_ind_slot2 (value a1_ind_v2)))
   (inh_slots ((((1749   1 ) a1_inh_slot1)
             (value a1_v1)(inh_condition default))
             (((1749  2 ) a1_inh_slot2)
              (value a1_v2)(inh_condition default))
             (((1749  3 ) a1_inh_slot3)
              (value a1_v3)(inh_condition default))
             (((1749  4 ) a1_inh_slot4)
              (value a1_v4)(inh_condition default)))))

 ((1249   p1)
  (id_slots ((offspring_no (value 3 ))
          (level_no (value 1 ))
          (parents (value none))
             (children (value ((181 p11)(281 p12)(381 p13))))))
   (ind_slots ((p1_ind_slot1  (value p1_ind_v1))))
   (inh_slots ((((1249   1 ) p1_inh_slot1)
             (value p1_inh_v1)(inh_condition default))
            (((1249  2 ) p1_inh_slot2)
             (value p1_inh_v2)(inh_condition default)))))

((1849   v1)
 (id_slots ((offspring_no (value 3 ))
         (level_no (value 1 ))
         (parents (value none))
            (children (value ((967 v11)(1967 v12)(981 v13)
                      (781 a12))))))
 (ind_slots ((v1_ind_slot1 (value v1_ind_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
            (value v1_inh_v1)(inh_condition default)))))

((249   z1)
 (id_slots ((offspring_no (value 3 ))
         (level_no (value 1 ))
         (parents (value none))
             (children (value ((1181 z11)(1281 z12)(1381 z13))))))
 (ind_slots ((z1_ind_slot1 (value z1_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
            (value z1_inh_v1)(inh_condition default))))) )

(((681   a11)
  (id_slots ((offspring_no (value 3 ))
          (level_no (value 2 ))
           (parents (value (1749 a1)))
             (children (value ((682 a21)(782 a22)(1968 v22))))))
   (ind_slots ((a11_ind_slot1 (value a11_ind_v1))))
   (inh_slots ((((1749   1 ) a1_inh_slot1)
             (value nil)(inh_condition default))
             (((1749  2 ) a1_inh_slot2)
              (value a11_inh_v1)(inh_condition override))
             (((1749  3 ) a1_inh_slot3)
              (value a11_inh_v2)(inh_condition union))
             (((1749  4 ) a1_inh_slot4)
              (value a11_inh_v3)(inh_condition intersection))
             (((681  1 ) a11_inh_slot1)
              (value a11_inh_v4)(inh_condition default)))))

((781   a12)
 (id_slots ((offspring_no (value 3 ))
         (level_no (value 2 ))
```

```
          (parents (value ((1749 a1)(1849 v1))))
          (children (value ((882 a23)(982 a24)(282 p22))))))
 (ind_slots ((a12_ind_slot1 (value a12_ind_v1))))
 (inh_slots ((((1749  1 ) a1_inh_slot1)
          (value nil)(inh_condition default))
          (((1749  2 ) a1_inh_slot2)
          (value a12_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
          (value a12_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
          (value a12_inh_v4)(inh_condition override))
          (((1849  1 ) v1_inh_slot1)
          (value a12_inh_v5)(inh_condition union))
          (((781  1 ) a12_inh_slot1)
          (value a12_inh_v6)(inh_condition default)))))

((881  a13)
 (id_slots ((offspring_no (value 4 ))
        (level_no (value 2 ))
        (parents (value (1749 a1)))
        (children (value ((1082 a25)(1182 a26)
                  (682 a21)(373 z32))))))
 (ind_slots ((a13_ind_slot1 (value a13_ind_v1))
          (a13_ind_slot2 (value a13_ind_v2))))
 (inh_slots ((((1749  1 ) a1_inh_slot1)
          (value a13_inh_v1)(inh_condition override))
          (((1749  2 ) a1_inh_slot2)
          (value a13_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
          (value a13_inh_v3)(inh_condition intersection))
          (((1749  4 ) a1_inh_slot4)
          (value nil)(inh_condition default))
          (((881  1 ) a13_inh_slot1)
          (value a13_inh_v5)(inh_condition default)))))

((181  p11)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 2 ))
        (parents (value (1249 p1)))
        (children (value ((182 p21)(282 p22))))))
 (ind_slots ((p11_ind_slot1 (value p11_ind_v1))))
 (inh_slots ((((1249  1 ) p1_inh_slot1)
          (value p11_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
          (value p11_inh_v2)(inh_condition override))
          (((181  1 ) p11_inh_slot1)
          (value p11_inh_v3)(inh_condition default)))))

((281  p12)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 2 ))
        (parents (value (1249 p1)))
        (children (value ((382 p23)(1372 z21))))))
 (ind_slots ((p12_ind_slot1 (value p12_ind_v1))))
 (inh_slots ((((1249  1 ) p1_inh_slot1)
          (value p12_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
          (value nil)(inh_condition default))
          (((281  1 ) p12_inh_slot1)
          (value p12_inh_v3)(inh_condition default)))))

((381  p13)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 2 ))
        (parents (value (1249 p1)))
        (children (value (482 p24)))))
 (ind_slots ((p13_ind_slot1 (value p13_ind_v1))))
 (inh_slots ((((1249  1 ) p1_inh_slot1)
          (value p13_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
          (value p13_inh_v2)(inh_condition default))
          (((381  1 ) p13_inh_slot1)
          (value p13_inh_v3)(inh_condition default)))))

((967  v11)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 2 ))
        (parents (value (1849 v1)))
        (children (value ((968 v21)(1968 v22))))))
 (ind_slots ((v11_ind_slot1 (value v11_v1))))
```

```
 (inh_slots ((((1849   1 ) v1_inh_slot1)
              (value nil)(inh_condition default))
              (((967  1 ) v11_inh_slot1)
               (value v11_inh_v1)(inh_condition default)))))

((1967   v12)
 (id_slots ((offspring_no (value 2 ))
            (level_no (value 2 ))
             (parents (value (1849 v1)))
               (children (value ((800 v23)(1968 v22))))))
 (ind_slots ((v12_ind_slot1  (value  v12_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
              (value v12_inh_v1)(inh_condition union))
              (((1967  1 ) v12_inh_slot1)
               (value v12_inh_v2)(inh_condition default)))))

((981   v13)
 (id_slots ((offspring_no (value 3 ))
            (level_no (value 2 ))
             (parents (value (1849 v1)))
               (children (value ((1800 v24)(1086 v25)(1382 z23))))))
 (ind_slots ((v13_ind_slot1  (value  v13_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
              (value v13_inh_v1)(inh_condition override))
              (((981  1 ) v13_inh_slot1)
               (value v13_inh_v2)(inh_condition default)))))

((1181   z11)
 (id_slots ((offspring_no (value 3 ))
            (level_no (value 2 ))
             (parents (value (249 z1)))
               (children (value ((1372 z21)(1282 z22)(1482 z24))))))
 (ind_slots ((z11_ind  (value  z11_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1181  1 ) z11_inh_slot1)
               (value z_inh_v1)(inh_condition default)))))

((1281   z12)
 (id_slots ((offspring_no (value 4 ))
            (level_no (value 2 ))
             (parents (value (249 z1)))
             (children (value ((1382 z23)
                               (1482 z24)(1582 z25)(1682 z26))))))
 (ind_slots ((z12_ind  (value  z12_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1281  1 ) z12_inh_slot1)
               (value z12_inh_v1)(inh_condition default)))))

((1381   z13)
 (id_slots ((offspring_no (value 3 ))
            (level_no (value 2 ))
             (parents (value (249 z1)))
               (children (value ((1782 z27)(1882 z28)(1982 z29))))))
 (ind_slots ((z13_ind  (value  z13_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1381  1 ) z13_inh_slot1)
               (value z13_inh_v1)(inh_condition default)))))  )


(((682   a21)
  (id_slots ((offspring_no (value 3 ))
             (level_no (value 3 ))
               (parents (value ((681 a11)(881 a13))))
                (children (value ((683 a31)(783 a32)(1283 v36))))))
  (ind_slots ((a21_ind_slot1  (value  a21_ind_v1))))
  (inh_slots ((((1749   1 ) a1_inh_slot1)
               (value nil)(inh_condition default))
               (((1749  2) a1_inh_slot2)
                (value a21_inh_v2)(inh_condition union))
               (((1749  3 ) a1_inh_slot3)
                (value a21_inh_v3)(inh_condition union))
               (((1749  4 ) a1_inh_slot4)
                (value a21_inh_v4)(inh_condition override))
               (((681 1) a11_inh_slot1)
                (value a21_inh_v5)(inh_condition union))
               (((881  1 ) a13_inh_slot1)
                (value a21_inh_v6)(inh_condition union))
```

```
          (((682 1) a21_inh_slot1)
           (value a21_inh_v7)(inh_condition default)))))

((782   a22)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
         (parents (value (681 a11)))
         (children (value (883 a33)))))
 (ind_slots ((a22_ind_slot1 (value a22_ind_v1))
          (a22_ind_slot2 (value a22_ind_v2))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  2) a1_inh_slot2)
            (value a22_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
            (value a22_inh_v3)(inh_condition union))
           (((1749  4 ) a1_inh_slot4)
            (value a22_inh_v4)(inh_condition union))
           (((782 1) a22_inh_slot1)
            (value a22_inh_v5)(inh_condition default)))))

((882   a23)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (781 a12)))
          (children (value ((783 a32)(983 a34))))))
 (ind_slots ((a23_ind_slot1 (value a23_ind_v1))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  2 ) a1_inh_slot2)
            (value a23_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
            (value a23_inh_v3)(inh_condition union))
           (((1749  4 ) a1_inh_slot4)
            (value a23_inh_v4)(inh_condition override))
           (((781  1 ) a12_inh_slot1)
            (value a23_inh_v5)(inh_condition intersection))
           (((1849  1 ) v1_inh_slot1)
            (value a23_inh_v6)(inh_condition union))
           (((882  1 ) a23_inh_slot1)
            (value a23_inh_v7)(inh_condition default))
           (((882  2 ) a23_inh_slot2)
            (value a23_inh_v8)(inh_condition default)))))

((982   a24)
 (id_slots ((offspring_no (value none))
        (level_no (value 3 ))
         (parents (value (781 a12)))
        (children (value none))))
 (ind_slots ((a24_ind_slot1 (value a24_ind_v1))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  2 ) a1_inh_slot2)
            (value a24_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
            (value a24_inh_v3)(inh_condition override))
           (((1749  4 ) a1_inh_slot4)
            (value a24_inh_v4)(inh_condition union))
           (((781  1 ) a12_inh_slot1)
            (value a24_inh_v5)(inh_condition union))
           (((1849  1 ) v1_inh_slot1)
            (value a24_inh_v6)(inh_condition union))
           (((982  1 ) a24_inh_slot1)
            (value a24_inh_v7)(inh_condition default)))))

((1082   a25)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (881 a13)))
          (children (value ((1083 a35)(1183 a36))))))
 (ind_slots ((a25_ind_slot1 (value a25_ind_v1))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  2 ) a1_inh_slot2)
            (value a25_inh_v2)(inh_condition override))
           (((1749  3 ) a1_inh_slot3)
            (value a25_inh_v3)(inh_condition union))
           (((1749  4 ) a1_inh_slot4)
            (value a25_inh_v4)(inh_condition union))
           (((881  1 ) a13_inh_slot1)
```

```
          (value a25_inh_v5)(inh_condition union))
          (((1082 ) a25_inh_slot1)
           (value a25_inh_v6)(inh_condition default)))))

((1182   a26)
 (id_slots ((offspring_no (value none))
        (level_no (value 3 ))
         (parents (value (881 a13)))
        (children (value none))))
 (ind_slots  ((a26_ind_slot1  (value  a26_ind_v1))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
          (value nil)(inh_condition default))
          (((1749  2 ) a1_inh_slot2)
           (value a26_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a26_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a26_inh_v4)(inh_condition override))
          (((881  1 ) a13_inh_slot1)
           (value a26_inh_v5)(inh_condition union))
          (((1182  1 ) a26_inh_slot1)
           (value a26_inh_v6)(inh_condition default)))))

((182   p21)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (181 p11)))
         (children (value ((183 p31)(283 p32)
                   (1363 p36)(783 a32))))))
 (ind_slots  ((p21_ind_slot1  (value  p21_ind_v1))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
          (value p21_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
           (value p21_inh_v2)(inh_condition union))
          (((181  1 ) p11_inh_slot1)
           (value p21_inh_v3)(inh_condition intersection))
          (((182  1 ) p21_inh_slot1)
           (value p21_inh_v4)(inh_condition default)))))

((282   p22)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
          (parents (value ((181 p11)(781 a12))))
         (children (value (583 p35)))))
 (ind_slots  ((p22_ind_slot1  (value  p22_ind_v1))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
          (value p22_inh_v1)(inh_condition override))
          (((1249  2 ) p1_inh_slot2)
           (value p22_inh_v2)(inh_condition union))
          (((181  1 ) p11_inh_slot1)
           (value p22_inh_v3)(inh_condition union))
          (((1749  1 ) a1_inh_slot1)
          (value nil)(inh_condition default))
          (((1749  2 ) a1_inh_slot2)
           (value p22_inh_v4)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value p22_inh_v5)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
          (value nil)(inh_condition default))
          (((1849  1 ) v1_inh_slot1)
           (value p22_inh_v6)(inh_condition union))
          (((781  1 ) a12_inh_slot1)
          (value nil)(inh_condition default))
          (((282  1 ) p22_inh_slot1)
           (value p22_inh_v7)(inh_condition default)))))

((382   p23)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
         (parents (value (281 p12)))
         (children (value (383 p33)))))
 (ind_slots  ((p23_ind_slot1  (value  p23_ind_v1))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
          (value p23_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
          (value nil)(inh_condition default))
          (((281  1 ) p12_inh_slot1)
           (value p23_inh_v2)(inh_condition union))
          (((382  1 ) p23_inh_slot1)
           (value p23_inh_v3)(inh_condition default)))))
```

```
((482   p24)
 (id_slots ((offspring_no (value 4 ))
        (level_no (value 3 ))
         (parents (value (381 p13)))
          (children (value ((483 p34)(583 p35)(1363 p36)
                    (1083 a35))))))
 (ind_slots ((p24_ind_slot1  (value  p24_ind_v1))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value p24_inh_v1)(inh_condition union))
           (((1249  2 ) p1_inh_slot2)
           (value p24_inh_v2)(inh_condition union))
           (((381  1 ) p13_inh_slot1)
            (value p24_inh_v3)(inh_condition intersection))
           (((482  1 ) p24_inh_slot1)
            (value p24_inh_v4)(inh_condition default)))))

((968   v21)
 (id_slots ((offspring_no (value 3 ))
        (level_no (value 3 ))
         (parents (value (967 v11)))
           (children (value ((969 v31)(1969 v32)(1087 v35))))))
 (ind_slots ((v21_ind_slot1  (value  v21_ind_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((967  1 ) v11_inh_slot1)
           (value nil)(inh_condition default))
           (((968  1 ) v21_inh_slot1)
            (value v21_inh_v1)(inh_condition default)))))

((1968   v22)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
           (parents (value ((967 v11)(1967 v12)(681 a11))))
           (children (value (801 v33)(1383 v37)))))
 (ind_slots ((v22_ind_slot1  (value  v22_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((1967  1 ) v12_inh_slot1)
           (value nil)(inh_condition default))
           (((1849  1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((967  1 ) v11_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  1 ) a1_inh_slot1)
            (value v22_inh_v1)(inh_condition override))
           (((1749  2 ) a1_inh_slot2)
            (value v22_inh_v2)(inh_condition override))
           (((1749  3 ) a1_inh_slot3)
            (value v22_inh_v3)(inh_condition override))
           (((1749  4 ) a1_inh_slot4)
            (value v22_inh_v4)(inh_condition union))
           (((681  1 ) a11_inh_slot1)
           (value nil)(inh_condition default))
           (((1968  1 ) v22_inh_slot1)
            (value v22_inh_v5)(inh_condition default)))))

((800   v23)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (1967 v12)))
          (children (value ((1801 v34)(1087 v35))))))
 (ind_slots ((v23_ind_slot1  (value  v23_ind_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((1967  1 ) v12_inh_slot1)
           (value v23_inh_v1)(inh_condition union))
           (((800  1 ) v23_inh_slot1)
            (value v23_inh_v2)(inh_condition default)))))

((1800   v24)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (981 v13)))
          (children (value ((1283 v36)(583 p35))))))
 (ind_slots ((v24_ind_slot1  (value  v24_ind_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
           (value v24_inh_v1)(inh_condition override))
           (((981  1 ) v13_inh_slot1)
           (value v24_inh_v2)(inh_condition union))
```

```
          (((1800  1 ) v24_inh_slot1)
           (value v24_inh_v3)(inh_condition default)))))

((1086   v25)
 (id_slots ((offspring_no (value 3 ))
        (level_no (value 3 ))
         (parents (value (981 v13)))
           (children (value ((1383 v37)(1483 v38)(383 p33))))))
 (ind_slots ((v25_ind_slot1  (value  v25_ind_v1))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
           (value v25_inh_v1)(inh_condition union))
           (((981  1 ) v13_inh_slot1)
            (value v25_inh_v2)(inh_condition intersection))
           (((1086  1 ) v25_inh_slot1)
            (value v25_inh_v3)(inh_condition default)))))

((1372   z21)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
           (parents (value ((1181 z11)(281 p12))))
          (children (value (1373 z31)))))
 (ind_slots ((z21_ind_slot1  (value  z21_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1181  1 ) z11_inh_slot1)
           (value nil)(inh_condition default))
           (((1249  1 ) p1_inh_slot1)
            (value z21_inh_v1)(inh_condition override))
           (((1249  2 ) p1_inh_slot2)
            (value z21_inh_v2)(inh_condition union))
           (((281  1 ) p12_inh_slot1)
            (value z21_inh_v3)(inh_condition union))
           (((1372  1 ) z21_inh_slot1)
            (value z21_inh_v4)(inh_condition default)))))

((1282   z22)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
          (parents (value (1181 z11)))
          (children (value (373 z32)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value z22_inh_v1)(inh_condition union))
           (((1181  1 ) z11_inh_slot1)
           (value z22_inh_v2)(inh_condition union))
           (((1282  1 ) z22_inh_slot1)
            (value z22_inh_v3)(inh_condition default)))))

((1382   z23)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
           (parents (value ((1281 z12)(981 v13))))
           (children (value ((1605 z33)(605 z34))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value z23_inh_v1)(inh_condition union))
           (((1281  1 ) z12_inh_slot1)
            (value z23_inh_v2)(inh_condition override))
           (((1849  1 ) v1_inh_slot1)
            (value z23_inh_v3)(inh_condition override))
           (((981  1 ) v13_inh_slot1)
           (value z23_inh_v4)(inh_condition union))
           (((1382  1 ) z23_inh_slot1)
            (value z23_inh_v5)(inh_condition default)))))

((1482   z24)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
           (parents (value ((1281 z12)(1181 z11))))
           (children (value ((1199 z310)(1383 v37))))))
 (ind_slots ((z24_ind_slot1  (value  z24_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value nil)(inh_condition default))
           (((1181  1 ) z11_inh_slot1)
           (value z24_inh_v1)(inh_condition union))
           (((1482  1 ) z24_inh_slot1)
            (value z24_inh_v2)(inh_condition default)))))
```

```
((1582   z25)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (1281 z12)))
          (children (value ((1583 z35)(1683 z36))))))
 (ind_slots ((z25_ind_slot1 (value z25_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value nil)(inh_condition default))
           (((1582  1 ) z25_inh_slot1)
            (value z25_inh_v1)(inh_condition default)))))

((1682   z26)
 (id_slots ((offspring_no (value none))
        (level_no (value 3 ))
         (parents (value (1281 z12)))
        (children (value none))))
 (ind_slots ((z26_ind_slot1 (value z26_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value z25_inh_v1)(inh_condition override))
           (((1281  1 ) z12_inh_slot1)
           (value z25_inh_v2)(inh_condition union))
           (((1682  1 ) z25_inh_slot1)
            (value z25_inh_v3)(inh_condition default)))))

((1782   z27)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (1381 z13)))
          (children (value ((1783 z37)(1883 z38))))))
 (ind_slots ((z27_ind_slot1 (value z27_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1782  1 ) z27_inh_slot1)
            (value z27_inh_v1)(inh_condition default)))))

((1882   z28)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 3 ))
         (parents (value (1381 z13)))
         (children (value (1983 z39)))))
 (ind_slots ((z28_ind_slot1 (value z28_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1882  1 ) z28_ind_slot1)
            (value z28_inh_v1)(inh_condition default)))))

((1982   z29)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 3 ))
         (parents (value (1381 z13)))
          (children (value ((1199 z310)(1432 z311))))))
 (ind_slots ((z29_ind_slot1 (value z29_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1982  1 ) z29_inh_slot1)
            (value z29_inh_v1)(inh_condition default)))))  )

(((683   a31)
  (id_slots ((offspring_no (value none))
        (level_no (value 4 ))
         (parents (value (682 a21)))
        (children (value none))))
  (ind_slots ((a31_ind_slot1 (value a31_ind_v1))))
  (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value a31_inh_v1)(inh_condition union))
           (((1749  2 ) a1_inh_slot2)
           (value a31_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
           (value nil)(inh_condition default))
           (((1749  4 ) a1_inh_slot4)
           (value a31_inh_v3)(inh_condition union))
           (((681 1) a11_inh_slot1)
```

```
          (value a31_inh_v4)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value a31_inh_v5)(inh_condition union))
          (((682 1) a21_inh_slot1)
           (value a31_inh_v6)(inh_condition union))
          (((683  1 ) a31_inh_slot1)
            (value a31_inh_v6)(inh_condition default)))))

((783   a32)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
           (parents (value ((682 a21)(882 a23)(182 p21))))
          (children (value ((684 a41)(784 a42))))))
 (ind_slots  ((a32_ind_slot1  (value  a32_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a32_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a32_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a32_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a32_inh_v4)(inh_condition union))
          (((681  1 ) a11_inh_slot1)
           (value a32_inh_v5)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
            (value a32_inh_v6)(inh_condition intersection))
          (((682  1 ) a21_inh_slot1)
           (value a32_inh_v7)(inh_condition union))
           (((1849 v1) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((781  1 ) a12_inh_slot1)
           (value a32_inh_v8)(inh_condition union))
          (((882  1 ) a23_inh_slot1)
           (value a32_inh_v9)(inh_condition union))
          (((881  2 ) a23_inh_slot2)
           (value a32_inh_v10)(inh_condition union))
          (((1249  1 ) p1_inh_slot1)
           (value a32_inh_v11)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
           (value a32_inh_v12)(inh_condition union))
          (((181  1 ) p11_inh_slot1)
            (value a32_inh_v13)(inh_condition intersection))
          (((182  1 ) p21_inh_slot1)
           (value a32_inh_v14)(inh_condition union))
          (((783  1 ) a32_inh_slot1)
            (value a32_inh_v15)(inh_condition default)))))

((883   a33)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (782 a22)))
          (children (value ((884 a43)(984 a44))))))
 (ind_slots  ((a33_ind_slot1  (value  a33_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a33_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a33_inh_v2)(inh_condition override))
          (((1749  3 ) a1_inh_slot3)
           (value a33_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a33_inh_v4)(inh_condition union))
          (((681  1 ) a11_inh_slot1)
            (value a33_inh_v5)(inh_condition intersection))
          (((782  1 ) a22_inh_slot1)
           (value a33_inh_v6)(inh_condition union))
          (((883  1 ) a33_inh_slot1)
           (value a33_inh_v7)(inh_condition override))
          (((883  2 ) a33_inh_slot2)
            (value a33_inh_v8)(inh_condition default)))))

((983   a34)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (882 a23)))
          (children (value ((1084 a45)(1184 a46))))))
 (ind_slots  ((a34_ind_slot1  (value  a34_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a34_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a34_inh_v2)(inh_condition union))
```

```
          (((1749  3 ) a1_inh_slot3)
           (value a34_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a34_inh_v4)(inh_condition union))
           (((1849 v1) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((781  1 ) a12_inh_slot1)
           (value a34_inh_v5)(inh_condition override))
          (((882  1 ) a23_inh_slot1)
           (value a34_inh_v6)(inh_condition override))
          (((882  2 ) a23_inh_slot2)
           (value a34_inh_v7)(inh_condition union))
          (((983  1 ) a34_inh_slot1)
           (value a34_inh_v8)(inh_condition default)))))

((1083   a35)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
          (parents (value ((1082 a25)(482 p24))))
         (children (value (1284 a47)))))
 (ind_slots  ((a35_ind_slot1  (value  a35_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a35_inh_v1)(inh_condition override))
          (((1749  2 ) a1_inh_slot2)
           (value a35_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a35_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a35_inh_v4)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value a35_inh_v5)(inh_condition union))
          (((1082  1 ) a25_inh_slot1)
           (value a35_inh_v6)(inh_condition union))
          (((482 1) p24_inh_slot1)
          (value nil)(inh_condition default))
          (((1083  1 ) a35_inh_slot1)
           (value a35_inh_v7)(inh_condition default)))))

((1183   a36)
 (id_slots ((offspring_no (value none))
        (level_no (value 4 ))
         (parents (value (1082 a25)))
        (children (value none))))
 (ind_slots  ((a36_ind_slot1  (value  a36_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a36_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a36_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a36_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a36_inh_v4)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value a36_inh_v5)(inh_condition union))
          (((1082  1 ) a25_inh_slot1)
           (value a36_inh_v6)(inh_condition union))
          (((1183  1 ) a36_inh_slot1)
           (value a36_inh_v7)(inh_condition default)))))

((183   p31)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
         (parents (value (182 p21)))
         (children (value (184 p41)))))
 (ind_slots ((nil (value nil))))
 (inh_slots  ((((1249   1 ) p1_inh_slot1)
           (value p31_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
          (value nil)(inh_condition default))
          (((182  1 ) p21_inh_slot1)
           (value p31_inh_v2)(inh_condition union))
          (((183  1 ) p31_inh_slot1)
           (value p31_inh_v3)(inh_condition default)))))

((283   p32)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (182 p21)))
          (children (value ((284 p42)(384 p43))))))
 (ind_slots ((nil (value nil))))
```

```
 (inh_slots ((((1249   1 ) p1_inh_slot1)
              (value nil)(inh_condition default))
              (((1249  2 ) p1_inh_slot2)
               (value p32_inh_v1)(inh_condition union))
              (((182  1 ) p21_inh_slot1)
               (value p32_inh_v2)(inh_condition union))
              (((283  1 ) p32_inh_v3)
               (value p32_v3)(inh_condition default)))))

((383   p33)
 (id_slots ((offspring_no (value 2 ))
            (level_no (value 4 ))
              (parents (value ((382 p23)(1086 v25))))
              (children (value ((484 p44)(584 p45))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1086   1 ) v25_inh_slot1)
              (value p33_inh_v1)(inh_condition union))
              (((1249  1 ) p1_inh_slot1)
              (value nil)(inh_condition default))
              (((1249  2 ) p1_inh_slot2)
              (value nil)(inh_condition default))
              (((281  1 ) p12_inh_slot1)
               (value p33_inh_v2)(inh_condition override))
              (((382  1 ) p23_inh_slot1)
               (value p33_inh_v3)(inh_condition union))
              (((383  1 ) p33_inh_slot1)
               (value p33_inh_v4)(inh_condition default)))))

((483   p34)
 (id_slots ((offspring_no (value 2 ))
            (level_no (value 4 ))
             (parents (value (482 p24)))
              (children (value ((284 p42)(970 v41))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
              (value nil)(inh_condition default))
              (((1249  2 ) p1_inh_slot2)
              (value nil)(inh_condition default))
              (((381  1 ) p13_inh_slot1)
               (value p34_inh_v1)(inh_condition override))
              (((482  1 ) p24_inh_slot1)
               (value p34_inh_v2)(inh_condition union))
              (((483  1 ) p34_inh_slot1)
               (value P34_inh_v3)(inh_condition default)))))

((583   p35)
 (id_slots ((offspring_no (value 3 ))
            (level_no (value 4 ))
               (parents (value ((482 p24)(282 p22)(1800 v24))))
               (children (value ((1364 p46)(364 p47)(1802 v44))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
              (value p35_inh_v1)(inh_condition override))
              (((1249  2 ) p1_inh_slot2)
               (value p35_inh_v2)(inh_condition union))
              (((381  1 ) p13_inh_slot1)
              (value nil)(inh_condition default))
              (((482  1 ) p24_inh_slot1)
               (value p35_inh_v3)(inh_condition union))
              (((781 1) a12_inh_slot1)
               (value p35_inh_v4)(inh_condition default))
              (((282  1 ) p22_inh_slot1)
               (value p35_inh_v5)(inh_condition union))
              (((981  1 ) v13_inh_slot1)
               (value p35_inh_v6)(inh_condition union))
              (((1800  1 ) v24_inh_slot1)
               (value p35_inh_v7)(inh_condition override))
              (((583  1 ) p35_inh_slot1)
               (value p35_inh_v8)(inh_condition default)))))


((1363   p36)
 (id_slots ((offspring_no (value 1 ))
            (level_no (value 4 ))
              (parents (value ((482 p24)(182 p21))))
             (children (value (584 p45)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
              (value p36_inh_v1)(inh_condition union))
              (((1249  2 ) p1_inh_slot2)
```

```
          (value p36_inh_v2)(inh_condition override))
         (((381  1 ) p13_inh_slot1)
         (value nil)(inh_condition default))
         (((482  1 ) p24_inh_slot1)
         (value nil)(inh_condition default))
         (((182  1 ) p21_inh_slot1)
          (value p36_inh_v3)(inh_condition intersection))
         (((1363  1 ) p36_inh_slot1)
          (value P36_inh_v4)(inh_condition default)))))

((969   v31)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (968 v21)))
          (children (value ((970 v41)(1970 v42))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((967  1 ) v11_inh_slot1)
          (value nil)(inh_condition default))
         (((968  1 ) v21)
          (value nil)(inh_condition default))
          (((969  1 ) v31_inh_slot1)
           (value v31_inh_v1)(inh_condition default)))))

((1969   v32)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
         (parents (value (968 v21)))
         (children (value (802 v43)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((967  1 ) v11_inh_slot1)
          (value nil)(inh_condition default))
         (((968  1 ) v21)
          (value nil)(inh_condition default))
          (((1969  1 ) v32_inh_slot1)
           (value v32_inh_v1)(inh_condition default)))))

((801   v33)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (1968 v22)))
          (children (value ((1802 v44)(1088 v45))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749   4 ) a1_inh_slot4)
          (value v33_inh_v1)(inh_condition union))
          (((681  1 ) a11_inh_slot1)
          (value v33_inh_v2)(inh_condition override))
          (((1967  1 ) v12_inh_slot1)
          (value nil)(inh_condition default))
          (((1849  1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((967  1 ) v11_inh_slot1)
          (value nil)(inh_condition default))
          (((1968  1 ) v22_inh_slot1)
          (value nil)(inh_condition default))
          (((801  1 ) v33_inh_slot1)
           (value v33_inh_v3)(inh_condition default)))))

((1801   v34)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
         (parents (value (800 v23)))
         (children (value (802 v43)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((1967  1 ) v12_inh_slot1)
          (value nil)(inh_condition default))
          (((800  1 ) v23_inh_slot1)
          (value nil)(inh_condition default))
          (((1801  1 ) v34_inh_slot1)
           (value v34_inh_v1)(inh_condition default)))))

((1087   v35)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
          (parents (value ((800 v23)(968 v21))))
```

```
          (children (value (802 v43)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1967   1 ) v12_inh_slot1)
           (value nil)(inh_condition default))
           (((800  1 ) v23_inh_slot1)
           (value nil)(inh_condition default))
           (((1849  1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((967  1 ) v11_inh_slot1)
           (value nil)(inh_condition default))
           (((968  1 ) v21)
           (value nil)(inh_condition default))
           (((1087  1 ) v35_inh_slot1)
            (value v35_inh_v1)(inh_condition default)))))

((1283   v36)
 (id_slots ((offspring_no (value 2 ))
         (level_no (value 4 ))
           (parents (value ((1800 v24)(682 a21))))
           (children (value ((88 v46)(1384 v47))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value v36_inh_v1)(inh_condition union))
           (((1749  2 ) a1_inh_slot2)
           (value v36_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
           (value v36_inh_v3)(inh_condition override))
           (((1749  4 ) a1_inh_slot4)
           (value v36_inh_v4)(inh_condition override))
           (((681  1 ) a11_inh_slot1)
            (value v36_inh_v5)(inh_condition intersection))
           (((881  1 ) a13_inh_slot1)
           (value v36_inh_v6)(inh_condition union))
           (((682  1 ) a21_inh_slot1)
           (value v36_inh_v7)(inh_condition override))
           (((981  1 ) v13_inh_slot1)
           (value nil)(inh_condition default))
           (((1800  1 ) v24_inh_slot1)
           (value nil)(inh_condition default))
           (((1283  1 ) v36_inh_slot1)
            (value v36_inh_v8)(inh_condition default)))))

((1383   v37)
 (id_slots ((offspring_no (value 1 ))
         (level_no (value 4 ))
            (parents (value ((1086 v25)(1968 v22)(1482 z24))))
           (children (value (802 v43)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1281   1 ) z12_inh_slot1)
           (value v37_inh_v1)(inh_condition union))
           (((249  1 ) z1_inh_slot1)
           (value v37_inh_v2)(inh_condition union))
           (((1181  1 ) z11_inh_slot1)
           (value v37_inh_v3)(inh_condition union))
           (((1482  1 ) z24_inh_slot1)
            (value v37_inh_v4)(inh_condition intersection))
           (((1086  1 ) v25_inh_slot1)
           (value nil)(inh_condition default))
           (((1749  4 ) a1_inh_slot4)
           (value v37_inh_v5)(inh_condition override))
           (((681  1 ) a11_inh_slot1)
           (value v37_inh_v6)(inh_condition union))
           (((1967  1 ) v12_inh_slot1)
           (value nil)(inh_condition default))
           (((1849  1 ) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((967  1 ) v11_inh_slot1)
           (value nil)(inh_condition default))
           (((1968  1 ) v22_inh_slot1)
           (value nil)(inh_condition default))
           (((1383  1 ) v37_inh_slot1)
            (value v37_inh_v7)(inh_condition default)))))

((1483   v38)
 (id_slots ((offspring_no (value none))
         (level_no (value 4 ))
           (parents (value (1086 v25)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1086   1 ) v25_inh_slot1)
```

```
          (value nil)(inh_condition default))
          (((1483  1 ) v38_inh_slot1)
           (value v38_inh_v1)(inh_condition default)))))

((1373   z31)
 (id_slots ((offspring_no (value 3 ))
        (level_no (value 4 ))
         (parents (value (1372 z21)))
           (children (value ((1374 z41)(374 z42)(1384 v47))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249  2 ) p1_inh_slot2)
           (value z31_inh_v1)(inh_condition union))
          (((281  1 ) p12_inh_slot1)
           (value z31_inh_v2)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1181  1 ) z11_inh_slot1)
          (value nil)(inh_condition default))
          (((1372  1 ) z21_inh_slot1)
          (value nil)(inh_condition default))
          (((1373  1 ) z31_inh_slot1)
           (value z31_inh_v3)(inh_condition default)))))

((373   z32)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
          (parents (value ((1282 z22)(881 a13))))
          (children (value ((1606 z43)(1484 z44))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749  2 ) a1_inh_slot2)
           (value z32_inh_v1)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value z32_inh_v2)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value z32_inh_v3)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value z32_inh_v4)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1181  1 ) z11_inh_slot1)
          (value nil)(inh_condition default))
          (((1282  1 ) z22_inh_slot1)
          (value nil)(inh_condition default))
          (((373  1 ) z32_Inh_slot1)
           (value z32_inh_v5)(inh_condition default)))))

((1605   z33)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
         (parents (value (1382 z23)))
         (children (value (1584 z45)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((981  1 ) v13_inh_slot1)
           (value z33_inh_v1)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1382  1 ) v23_inh_slot1)
           (value z33_inh_v2)(inh_condition union))
          (((1605  1 ) z33_inh_slot1)
           (value z33_inh_v3)(inh_condition default)))))

((605   z34)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (1382 z23)))
           (children (value ((374 z42)(1606 z43))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((981  1 ) v13_inh_slot1)
           (value z34_inh_v1)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1382  1 ) v23_inh_slot1)
           (value z34_inh_v2)(inh_condition intersection))
          (((605  1 ) z34_inh_slot1)
           (value z34_inh_v3)(inh_condition default)))))

((1583   z35)
 (id_slots ((offspring_no (value 1 ))
        (level_no (value 4 ))
         (parents (value (1582 z25)))
```

```
          (children (value (1202 z412)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value nil)(inh_condition default))
           (((1582  1 ) z25_inh_slot1)
           (value nil)(inh_condition default))
           (((1583  1 ) z35_inh_slot1)
            (value z35_inh_v1)(inh_condition default)))))

((1683   z36)
 (id_slots ((offspring_no (value 2 ))
         (level_no (value 4 ))
          (parents (value (1582 z25)))
           (children (value ((1684 z46)(1784 z47))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value nil)(inh_condition default))
           (((1582  1 ) z25_inh_slot1)
           (value nil)(inh_condition default))
           (((1683  1 ) z36_Inh_slot1)
            (value z36_inh_v1)(inh_condition default)))))

((1783   z37)
 (id_slots ((offspring_no (value 1 ))
         (level_no (value 4 ))
          (parents (value (1782 z27)))
          (children (value (1884 z48)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1782  1 ) z27_inh_slot1)
           (value nil)(inh_condition default))
           (((1783  1 ) z37_inh_slot1)
            (value z37_inh_v1)(inh_condition default)))))

((1883   z38)
 (id_slots ((offspring_no (value 1 ))
         (level_no (value 4 ))
          (parents (value (1782 z27)))
          (children (value (1984 z49)))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1782  1 ) z27_inh_slot1)
           (value nil)(inh_condition default))
           (((1883  1 ) z38_inh_slot1)
            (value z38_inh_v1)(inh_condition default)))))

((1983   z39)
 (id_slots ((offspring_no (value 2 ))
         (level_no (value 4 ))
          (parents (value (1882 z28)))
           (children (value ((1200 z410)(1201 z411))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1882  1 ) z28_ind_slot1)
           (value nil)(inh_condition default))
           (((1983  1 ) z39_inh_slot1)
            (value z39_inh_v1)(inh_condition default)))))

((1199   z310)
 (id_slots ((offspring_no (value 2 ))
         (level_no (value 4 ))
           (parents (value ((1982 z29)(1482 z24))))
           (children (value ((1584 z45)(1364 p46))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1381   1 ) z13_inh_slot1)
           (value z310_inh_v1)(inh_condition union))
           (((1982  1 ) z29_inh_slot1)
```

```
          (value nil)(inh_condition default))
          (((1281  1 ) z12_inh_slot1)
           (value z310_inh_v2)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1181  1 ) z11_inh_slot1)
           (value z310_inh_v3)(inh_condition union))
          (((1482  1 ) z24_inh_slot1)
           (value z310_inh_v4)(inh_condition union))
          (((1199  1 ) z310_inh_slot1)
           (value z310_inh_v5)(inh_condition default)))))

((1432   z311)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
         (parents (value (1982 z29)))
          (children (value ((1202 z412)(1203 z413))))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1381  1 ) z13_inh_slot1)
          (value nil)(inh_condition default))
          (((1982  1 ) z29_inh_slot1)
          (value nil)(inh_condition default))
          (((1432  1 ) z311_inh_slot1)
           (value z311_inh_v1)(inh_condition default)))))  )

(((684   a41)
  (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (783 a32)))
         (children (value none))))
  (ind_slots ((a41_ind_slot1  (value  a41_ind_v1))))
  (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value a41_inh_v1)(inh_condition union))
           (((1749  2 ) a1_inh_slot2)
           (value a41_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
           (value a41_inh_v3)(inh_condition union))
           (((1749  4 ) a1_inh_slot4)
           (value a41_inh_v4)(inh_condition union))
           (((681  1 ) a11_inh_slot1)
           (value a41_inh_v5)(inh_condition union))
           (((881  1 ) a13_inh_slot1)
           (value a41_inh_v6)(inh_condition union))
           (((682  1 ) a21_inh_slot1)
           (value a41_inh_v7)(inh_condition union))
           (((1849 1) v1_inh_slot1)
           (value nil)(inh_condition default))
           (((781  1 ) a12_inh_slot1)
           (value a41_inh_v8)(inh_condition default))
           (((882  1 ) a23_inh_slot1)
           (value a41_inh_v9)(inh_condition default))
           (((882  2 ) a23_inh_slot2)
           (value a41_inh_v10)(inh_condition default))
           (((1249 1) p1_inh_slot1)
           (value nil)(inh_condition default))
           (((181 1) p11_inh_slot1)
           (value nil)(inh_condition default))
           (((182 1) p21_inh_slot1)
           (value nil)(inh_condition default))
           (((783  1 ) a32_inh_slot1)
           (value a41_inh_v11)(inh_condition default))
           (((684  1 ) a41_inh_slot1)
           (value a41_inh_v12)(inh_condition default)))))

((784   a42)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((783 a32)(1373 z31))))
        (children (value none))))
 (ind_slots ((a42_ind_slot1  (value  a42_ind_v1))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value a42_inh_v1)(inh_condition union))
           (((1749  2 ) a1_inh_slot2)
           (value a42_inh_v2)(inh_condition union))
           (((1749  3 ) a1_inh_slot3)
           (value a42_inh_v3)(inh_condition union))
           (((1749  4 ) a1_inh_slot4)
           (value a42_inh_v4)(inh_condition union))
```

```
            (((681  1 ) a11_inh_slot1)
             (value a42_inh_v5)(inh_condition union))
            (((881  1 ) a13_inh_slot1)
             (value a42_inh_v6)(inh_condition union))
            (((682  1 ) a21_inh_slot1)
             (value a42_inh_v7)(inh_condition union))
             (((1849 1) v1_inh_slot1)
            (value nil)(inh_condition union))
            (((781  1 ) a12_inh_slot1)
             (value a42_inh_v8)(inh_condition union))
            (((882  1 ) a23_inh_slot1)
             (value a42_inh_v9)(inh_condition union))
            (((881  2 ) a23_inh_slot2)
             (value a42_inh_v10)(inh_condition union))
            (((1249 1) p1_inh_slot1)
             (value nil)(inh_condition default))
              (((181 1) p11_inh_slot1)
             (value nil)(inh_condition default))
              (((182 1) p21_inh_slot1)
             (value nil)(inh_condition default))
            (((783  1 ) a32_inh_slot1)
             (value a42_inh_v11)(inh_condition union))
            (((249 1) z1_inh_slot1)
            (value nil)(inh_condition default))
             (((1181 1) z11_inh_slot1)
            (value nil)(inh_condition default))
             (((1372 1) z21_inh_slot1)
            (value nil)(inh_condition default))
             (((1373 1) z31_inh_slot1)
            (value nil)(inh_condition default))
            (((281 1) p12_inh_slot1)
            (value nil)(inh_condition default))
            (((784  1 ) a42_inh_slot1)
              (value a42_inh_v12)(inh_condition default)))))

((884   a43)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (883 a33)))
         (children (value none))))
 (ind_slots ((a43_ind_slot1 (value a43_ind_v1))))
 (inh_slots ((((1749  1 ) a1_inh_slot1)
            (value a43_inh_v1)(inh_condition union))
            (((1749  2 ) a1_inh_slot2)
             (value a43_inh_v2)(inh_condition union))
            (((1749  3 ) a1_inh_slot3)
             (value a43_inh_v3)(inh_condition union))
            (((1749  4 ) a1_inh_slot4)
             (value a43_inh_v4)(inh_condition union))
            (((681  1 ) a11_inh_slot1)
             (value a43_inh_v5)(inh_condition union))
            (((782  1 ) a22_inh_slot1)
             (value a43_inh_v6)(inh_condition union))
            (((883  1 ) a33_inh_slot1)
             (value a43_inh_v7)(inh_condition union))
            (((883  2 ) a33_inh_slot2)
             (value a43_inh_v8)(inh_condition union)))))

((984   a44)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (883 a33)))
         (children (value none))))
 (ind_slots ((a44_ind_slot1 (value a44_ind_v1))))
 (inh_slots ((((1749  1 ) a1_inh_slot1)
            (value a44_inh_v1)(inh_condition union))
            (((1749  2 ) a1_inh_slot2)
             (value a44_inh_v2)(inh_condition union))
            (((1749  3 ) a1_inh_slot3)
             (value a44_inh_v3)(inh_condition union))
            (((1749  4 ) a1_inh_slot4)
             (value a44_inh_v4)(inh_condition union))
            (((681  1 ) a11_inh_slot1)
             (value a44_inh_v5)(inh_condition union))
            (((782  1 ) a22_inh_slot1)
             (value a44_inh_v6)(inh_condition union))
            (((883  1 ) a33_inh_slot1)
             (value a44_inh_v7)(inh_condition union))
            (((883  2 ) a33_inh_slot2)
             (value a44_inh_v8)(inh_condition union))
```

```
          (((984  1 ) a44_inh_slot2)
           (value a44_inh_v9)(inh_condition default)))))

((1084   a45)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (983 a34)))
        (children (value none))))
 (ind_slots  ((a45_ind_slot1  (value  a45_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a45_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a45_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a45_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a45_inh_v4)(inh_condition union))
          (((1849 1) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((781  1 ) a12_inh_slot1)
           (value a45_inh_v5)(inh_condition union))
          (((882  1 ) a23_inh_slot1)
           (value a45_inh_v6)(inh_condition union))
          (((882  2 ) a23_inh_slot2)
           (value a45_inh_v7)(inh_condition union))
          (((983  1 ) a34_inh_slot1)
           (value a45_inh_v8)(inh_condition default)))))

((1184   a46)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (983 a34)))
        (children (value none))))
 (ind_slots  ((a46_ind_slot1  (value  a46_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a46_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a46_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a46_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a46_inh_v4)(inh_condition union))
          (((1849 1) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((781  1 ) a12_inh_slot1)
           (value a46_inh_v5)(inh_condition union))
          (((882  1 ) a23_inh_slot1)
           (value a46_inh_v6)(inh_condition union))
          (((882  2 ) a23_inh_slot2)
           (value a46_inh_v7)(inh_condition union))
          (((983  1 ) a34_inh_slot1)
           (value a46_inh_v8)(inh_condition default)))))

((1284   a47)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (1083 a35)))
        (children (value none))))
 (ind_slots  ((a47_ind_slot1  (value  a47_ind_v1))))
 (inh_slots  ((((1749   1 ) a1_inh_slot1)
           (value a47_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value a47_inh_v2)(inh_condition union))
          (((1749  3 ) a1_inh_slot3)
           (value a47_inh_v3)(inh_condition union))
          (((1749  4 ) a1_inh_slot4)
           (value a47_inh_v4)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value a47_inh_v5)(inh_condition union))
          (((1082  1 ) a25_inh_slot1)
           (value a47_inh_v6)(inh_condition union))
          (((1249 1)p1_inh_slot1)
          (value nil)(inh_condition default))
          (((381 1) p13_inh_slot1)
          (value nil)(inh_condition default))
          (((482 1) p24_inh_slot1)
          (value nil)(inh_condition default))
          (((1083  1 ) a35_inh_slot1)
           (value a47_inh_v7)(inh_condition default)))))  )
```

```
(((184   p41)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (183 p31)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value nil)(inh_condition union))
           (((1249  2 ) p1_inh_slot2)
            (value p41_inh_v1)(inh_condition union))
           (((182  1 ) p21_inh_slot1)
            (value p41_inh_v2)(inh_condition union))
           (((183  1 ) p31_inh_slot1)
           (value nil)(inh_condition union))
           (((184  1 ) p41_inh_slot1)
            (value p41_inh_v1)(inh_condition default)))))

((284   p42)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
           (parents (value ((283 p32)(483 p34))))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value nil)(inh_condition default))
           (((1249  2 ) p1_inh_slot2)
           (value nil)(inh_condition default))
           (((482  1 ) p24_inh_slot1)
           (value nil)(inh_condition default))
           (((483  1 ) p34_inh_slot1)
           (value nil)(inh_condition default))
           (((182  1 ) p21_inh_slot1)
           (value nil)(inh_condition default))
           (((283  1 ) p32_inh_v3)
           (value nil)(inh_condition default))
           (((284  1 ) p42_inh_slot1)
            (value p42_inh_v1)(inh_condition default)))))

((384   p43)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (283 p32)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value nil)(inh_condition default))
           (((1249  2 ) p1_inh_slot2)
           (value nil)(inh_condition default))
           (((182  1 ) p21_inh_slot1)
           (value nil)(inh_condition default))
           (((283  1 ) p32_inh_v3)
           (value nil)(inh_condition default))
           (((384  1 ) p43_inh_slot1)
            (value p43_inh_v1)(inh_condition default)))))

((484   p44)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (383 p33)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value nil)(inh_condition default))
           (((1249  2 ) p1_inh_slot2)
           (value nil)(inh_condition default))
           (((382  1 ) p23_inh_slot1)
           (value nil)(inh_condition default))
           (((1849  1 ) v1_inh_slot1)
           (value p44_inh_v1)(inh_condition union))
           (((981  1 ) v13_inh_slot1)
           (value p44_inh_v2)(inh_condition union))
           (((1086  1 ) v25_inh_slot1)
           (value p44_inh_v3)(inh_condition union))
           (((383  1 ) p33_inh_slot1)
           (value nil)(inh_condition default))
           (((484  1 ) p44_inh_slot1)
            (value p44_inh_v4)(inh_condition default)))))

((584   p45)
```

```
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
           (parents (value ((383 p33)(1363 p36))))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
            (value p45_inh_v1)(inh_condition union))
            (((1249  2 ) p1_inh_slot2)
            (value p45_inh_v2)(inh_condition union))
            (((382  1 ) p23_inh_slot1)
            (value p45_inh_v3)(inh_condition unin))
            (((1086  1 ) v25_inh_slot1)
            (value p45_inh_v4)(inh_condition union))
            (((383  1 ) p33_inh_slot1)
            (value p45_inh_v5)(inh_condition union))
            (((381  1 ) p13_inh_slot1)
             (value p45_inh_v6)(inh_condition intersection))
            (((482  1 ) p24_inh_slot1)
            (value nil)(inh_condition default))
            (((182  1 ) p21_inh_slot1)
            (value p45_inh_v7)(inh_condition union))
            (((1363  1 ) p36_inh_slot1)
            (value p45_inh_v8)(inh_condition union))
            (((584  1 ) p45_inh_slot1)
             (value p45_inh_v9)(inh_condition default)))))

((1364   p46)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
           (parents (value ((583 p35)(1199 z310))))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
            (value nil)(inh_condition default))
            (((1249  2 ) p1_inh_slot2)
            (value nil)(inh_condition default))
            (((381  1 ) p13_inh_slot1)
            (value nil)(inh_condition default))
            (((482  1 ) p24_inh_slot1)
            (value nil)(inh_condition default))
            (((1849  1 ) v1_inh_slot1)
            (value p46_inh_v1)(inh_condition union))
            (((981  1 ) v13_inh_slot1)
            (value p46_inh_v2)(inh_condition union))
            (((1800  1 ) v24_inh_slot1)
            (value p46_inh_v3)(inh_condition union))
            (((781 1) a12_inh_slot1)
            (value p46_inh_v4)(inh_condition union))
            (((282  1 ) p22_inh_slot1)
            (value p46_inh_v5)(inh_condition union))
            (((583  1 ) p35_inh_slot1)
            (value nil)(inh_condition default))
            (((249  1 ) z1_inh_slot1)
            (value p46_inh_v6)(inh_condition union))
            (((1381  1 ) z13_inh_slot1)
            (value p46_inh_v7)(inh_condition union))
            (((1982  1 ) z29_inh_slot1)
            (value p46_inh_v8)(inh_condition union))
            (((1181  1 ) z11_inh_slot1)
            (value p46_inh_v9)(inh_condition override))
            (((1281  1 ) z12_inh_slot1)
            (value p46_inh_v10)(inh_condition union))
            (((1482  1 ) z24_inh_slot1)
             (value p46_inh_v11)(inh_condition override))
            (((1199  1 ) z310_inh_slot1)
            (value p46_inh_v12)(inh_condition union))
            (((1364  1 ) p46_inh_slot1)
             (value p46_inh_v13)(inh_condition default)))))

((364   p47)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (583 p35)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
            (value nil)(inh_condition default))
            (((1249  2 ) p1_inh_slot2)
            (value nil)(inh_condition default))
            (((381  1 ) p13_inh_slot1)
```

```
          (value nil)(inh_condition default))
          (((482  1 ) p24_inh_slot1)
          (value nil)(inh_condition default))
          (((1849  1 ) v1_inh_slot1)
           (value p47_inh_v1)(inh_condition union))
          (((981  1 ) v13_inh_slot1)
           (value p47_inh_v2)(inh_condition union))
          (((1800  1 ) v24_inh_slot1)
           (value p47_inh_v3)(inh_condition union))
          (((781 1) a12_inh_slot1)
           (value p47_inh_v4)(inh_condition union))
          (((282  1 ) p22_inh_slot1)
           (value p47_inh_v5)(inh_condition union))
          (((583  1 ) p35_inh_slot1)
          (value nil)(inh_condition default))
          (((364  1 ) p47_inh_slot1)
            (value p47_inh_v6)(inh_condition default))))))


(((970   v41)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((969 v31)(483 p34))))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   1 ) p1_inh_slot1)
           (value p41_inh_v1)(inh_condition union))
          (((1249  2 ) p1_inh_slot2)
           (value v41_inh_v2)(inh_condition union))
          (((482  1 ) p24_inh_slot1)
           (value v41_inh_v3)(inh_condition override))
          (((483  1 ) p34_inh_slot1)
           (value v41_inh_v4)(inh_condition override))
          (((1849  1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((967  1 ) v11_inh_slot1)
          (value nil)(inh_condition default))
          (((968  1 ) v21_inh_slot1)
          (value nil)(inh_condition default))
          (((969  1 ) v31_inh_slot1)
          (value nil)(inh_condition default))
          (((970  1 ) v41_inh_slot1)
           (value v41_inh_v5)(inh_condition default)))))

((1970   v42)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (969 v31)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1849   1 ) v1_inh_slot1)
          (value nil)(inh_condition default))
          (((967  1 ) v11_inh_slot1)
          (value nil)(inh_condition default))
          (((968  1 ) v21_inh_slot1)
          (value nil)(inh_condition default))
          (((969  1 ) v31_inh_slot1)
          (value nil)(inh_condition default))
          (((1970  1 ) v42_inh_slot1)
           (value v42_inh_v1)(inh_condition default)))))

((802   v43)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((1969 v32)(1801 v34)
                    (1087 v35)(1383 v37))))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1281   1 ) z12_inh_slot1)
           (value v43_inh_v1)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
           (value v43_inh_v2)(inh_condition override))
          (((1181  1 ) z11_inh_slot1)
           (value v43_inh_v3)(inh_condition union))
          (((1482  1 ) z24_inh_slot1)
           (value v43_inh_v4)(inh_condition union))
          (((1086  1 ) v25_inh_slot1)
          (value nil)(inh_condition default))
          (((681  1 ) a11_inh_slot1)
           (value v43_inh_v5)(inh_condition union))
```

```
            (((1968  1 ) v22_inh_slot1)
            (value nil)(inh_condition default))
            (((1383  1 ) v37_inh_slot1)
            (value nil)(inh_condition default))
            (((1087  1 ) v35_inh_slot1)
            (value nil)(inh_condition default))
            (((1967  1 ) v12_inh_slot1)
            (value nil)(inh_condition default))
            (((800  1 ) v23_inh_slot1)
            (value nil)(inh_condition default))
            (((1801  1 ) v34_inh_slot1)
            (value nil)(inh_condition default))
            (((1849  1 ) v1_inh_slot1)
            (value nil)(inh_condition default))
            (((967  1 ) v11_inh_slot1)
            (value nil)(inh_condition default))
            (((968  1 ) v21_inh_slot1)
            (value nil)(inh_condition default))
            (((1969  1 ) v32_inh_slot1)
            (value nil)(inh_condition default))
            (((802  1 ) v43_inh_slot1)
             (value v43_inh_v6)(inh_condition default)))))

((1802   v44)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((801 v33)(583 p35))))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((981   1 ) v13_inh_slot1)
            (value nil)(inh_condition default))
            (((1249  1 ) p1_inh_slot1)
             (value v44_inh_v1)(inh_condition union))
            (((1249  2 ) p1_inh_slot2)
             (value v44_inh_v2)(inh_condition union))
            (((381  1 ) p13_inh_slot1)
             (value v44_inh_v3)(inh_condition override))
            (((482  1 ) p24_inh_slot1)
             (value v44_inh_v4)(inh_condition union))
            (((781 1) a12_inh_slot1)
             (value v44_inh_v6)(inh_condition union))
            (((282  1 ) p22_inh_slot1)
             (value v44_inh_v7)(inh_condition union))
            (((583  1 ) p35_inh_slot1)
             (value v44_inh_v8)(inh_condition union))
            (((1749  4 ) a1_inh_slot4)
             (value v44_inh_v9)(inh_condition union))
            (((1967  1 ) v12_inh_slot1)
            (value nil)(inh_condition default))
            (((1849  1 ) v1_inh_slot1)
            (value nil)(inh_condition default))
            (((967  1 ) v11_inh_slot1)
            (value nil)(inh_condition default))
            (((1968  1 ) v22_inh_slot1)
            (value nil)(inh_condition default))
            (((801  1 ) v33_inh_slot1)
            (value nil)(inh_condition default))
            (((1802  1 ) v44_inh_slot1)
             (value v44_inh_v10)(inh_condition default)))))

((1088   v45)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value (801 v33)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749   4 ) a1_inh_slot4)
             (value v45_inh_v1)(inh_condition union))
            (((1967  1 ) v12_inh_slot1)
            (value nil)(inh_condition override))
            (((1849  1 ) v1_inh_slot1)
            (value nil)(inh_condition default))
            (((967  1 ) v11_inh_slot1)
            (value nil)(inh_condition default))
            (((1968  1 ) v22_inh_slot1)
            (value nil)(inh_condition default))
            (((801  1 ) v33_inh_slot1)
            (value nil)(inh_condition default))
            (((1088  1 ) v45_inh_slot1)
             (value v45_inh_v2)(inh_condition default)))))
```

```
((88   v46)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (1283 v36)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749   1 ) a1_inh_slot1)
           (value v46_inh_v1)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value v46_inh_v2)(inh_condition union))
          (((681  1 ) a11_inh_slot1)
           (value v46_inh_v3)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value v46_inh_v4)(inh_condition union))
          (((981  1 ) v13_inh_slot1)
          (value nil)(inh_condition default))
          (((1800  1 ) v24_inh_slot1)
          (value nil)(inh_condition default))
          (((1283  1 ) v36_inh_slot1)
           (value v46_inh_v5)(inh_condition override))
          (((88  1 ) v46_inh_slot1)
           (value v46_inh_v6)(inh_condition default)))))

((1384   v47)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((1283 v36)(1373 z31))))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   2 ) p1_inh_slot2)
           (value v47_inh_v1)(inh_condition union))
          (((281  1 ) p12_inh_slot1)
           (value v47_inh_v2)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
           (value v47_inh_v3)(inh_condition override))
          (((1181  1 ) z11_inh_slot1)
           (value v47_inh_v4)(inh_condition union))
          (((1372  1 ) z21_inh_slot1)
           (value v47_inh_v5)(inh_condition union))
          (((1373  1 ) z31_inh_slot1)
           (value v47_inh_v6)(inh_condition union))
          (((1749  1 ) a1_inh_slot1)
           (value v47_inh_v7)(inh_condition union))
          (((1749  2 ) a1_inh_slot2)
           (value v47_inh_v8)(inh_condition union))
          (((681  1 ) a11_inh_slot1)
           (value v47_inh_v9)(inh_condition union))
          (((881  1 ) a13_inh_slot1)
           (value v47_inh_v10)(inh_condition union))
          (((981  1 ) v13_inh_slot1)
          (value nil)(inh_condition default))
          (((1800  1 ) v24_inh_slot1)
          (value nil)(inh_condition default))
          (((1283  1 ) v36_inh_slot1)
          (value nil)(inh_condition default))
          (((1384  1 ) v47_inh_slot1)
           (value v47_inh_v11)(inh_condition default)))))  )


(((1374   z41)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
         (parents (value (1373 z31)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1249   2 ) p1_inh_slot2)
           (value z41_inh_v1)(inh_condition union))
          (((249  1 ) z1_inh_slot1)
          (value nil)(inh_condition default))
          (((1181  1 ) z11_inh_slot1)
          (value nil)(inh_condition default))
          (((1372  1 ) z21_inh_slot1)
          (value nil)(inh_condition default))
          (((1373  1 ) z31_inh_slot2)
          (value nil)(inh_condition default))
          (((1374  1 ) z41_inh_slot1)
           (value z41_inh_v2)(inh_condition default)))))

((374   z42)
```

```
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
           (parents (value ((1373 z31)(605 z34))))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((981   1 ) v13_inh_slot1)
            (value z42_inh_v1)(inh_condition union))
            (((1382  1 ) v23_inh_slot1)
            (value z42_inh_v2)(inh_condition union))
            (((605  1 ) z34_inh_slot1)
            (value nil)(inh_condition default))
            (((1249  2 ) p1_inh_slot2)
             (value z42_inh_v3)(inh_condition intersection))
            (((281  1 ) p12_inh_slot1)
             (value z42_inh_v4)(inh_condition override))
            (((249  1 ) z1_inh_slot1)
            (value nil)(inh_condition default))
            (((1181  1 ) z11_inh_slot1)
            (value nil)(inh_condition default))
            (((1372  1 ) z21_inh_slot1)
            (value nil)(inh_condition default))
            (((1373  1 ) z31_inh_slot1)
            (value nil)(inh_condition default))
            (((374  1 ) z42_inh_slot1)
             (value z42_inh_v5)(inh_condition default)))))

((1606   z43)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
           (parents (value ((373 z32)(605 z34))))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((981   1 ) v13_inh_slot1)
            (value z43_inh_v1)(inh_condition union))
            (((1382  1 ) z23_inh_slot1)
            (value z43_inh_v2)(inh_condition union))
            (((605  1 ) z34_inh_slot1)
            (value nil)(inh_condition default))
            (((1749  2 ) a1_inh_slot2)
             (value v43_inh_v3)(inh_condition override))
            (((1749  3 ) a1_inh_slot3)
             (value z43_inh_v4)(inh_condition union))
            (((1749  4 ) a1_inh_slot4)
             (value z43_inh_v5)(inh_condition override))
            (((881  1 ) a13_inh_slot1)
             (value z43_inh_v6)(inh_condition union))
            (((249  1 ) z1_inh_slot1)
            (value nil)(inh_condition default))
            (((1181  1 ) z11_inh_slot1)
            (value nil)(inh_condition default))
            (((1282  1 ) z22_inh_slot1)
            (value nil)(inh_condition default))
            (((373  1 ) z32_Inh_slot1)
             (value z43_inh_v7)(inh_condition union))
            (((1606  1 ) z43_inh_slot1)
             (value z43_inh_v8)(inh_condition default)))))

((1484   z44)
 (id_slots ((offspring_no (value none))
         (level_no (value 5 ))
          (parents (value (373 z32)))
         (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1749   2 ) a1_inh_slot2)
            (value z44_inh_v1)(inh_condition union))
            (((1749  3 ) a1_inh_slot3)
             (value z44_inh_v2)(inh_condition union))
            (((1749  4 ) a1_inh_slot4)
             (value z44_inh_v3)(inh_condition override))
            (((881  1 ) a13_inh_slot1)
             (value z44_inh_v4)(inh_condition union))
            (((249  1 ) z1_inh_slot1)
            (value nil)(inh_condition default))
            (((1181  1 ) z11_inh_slot1)
            (value nil)(inh_condition default))
            (((1282  1 ) z22_inh_slot1)
            (value nil)(inh_condition default))
            (((373  1 ) z32_Inh_slot1)
             (value z44_inh_5)(inh_condition override))
            (((1484  1 ) z44_inh_slot1)
```

```
            (value z44_inh_v6)(inh_condition default)))))

((1584   z45)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value ((1605 z33)(1199 z310))))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1381   1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1982  1 ) z29_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value z45_inh_v1)(inh_condition union))
           (((1181  1 ) z11_inh_slot1)
           (value nil)(inh_condition default))
           (((1482  1 ) z24_inh_slot1)
           (value z45_inh_v2)(inh_condition override))
           (((1199  1 ) z310_inh_slot1)
           (value nil)(inh_condition default))
           (((981  1 ) v13_inh_slot1)
           (value z45_inh_v3)(inh_condition union))
           (((249  1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1382  1 ) z23_inh_slot1)
           (value z45_inh_v4)(inh_condition override))
           (((1605  1 ) z33_inh_slot1)
           (value nil)(inh_condition default))
           (((1584  1 ) z45_inh_slot1)
           (value z45_inh_v5)(inh_condition default)))))

((1684   z46)
 (id_slots ((offspring_no (value 2 ))
        (level_no (value 4 ))
          (parents (value (1683 z36)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((1683   1 ) z36_inh_slot1)
           (value z36_inh_v1)(inh_condition default)))))

((1784   z47)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value (1683 z36)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value nil)(inh_condition default))
           (((1281  1 ) z12_inh_slot1)
           (value nil)(inh_condition default))
           (((1582  1 ) z25_inh_slot1)
           (value nil)(inh_condition default))
           (((1683  1 ) z36_Inh_slot1)
           (value nil)(inh_condition default))
           (((1784  1 ) z47_inh_slot1)
           (value z47_inh_v1)(inh_condition default)))))

((1884   z48)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value (1783 z37)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
           (value z48_inh_v1)(inh_condition override))
           (((1381  1 ) z13_inh_slot1)
           (value nil)(inh_condition default))
           (((1782  1 ) z27_inh_slot1)
           (value nil)(inh_condition default))
           (((1783  1 ) z37_inh_slot1)
           (value nil)(inh_condition default))
           (((1884  1 ) z48_inh_slot1)
           (value z48_inh_v2)(inh_condition default)))))

((1984   z49)
 (id_slots ((offspring_no (value none))
        (level_no (value 5 ))
          (parents (value (1883 z38)))
        (children (value none))))
 (ind_slots ((nil (value nil))))
```

```
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1381  1 ) z13_inh_slot1)
              (value nil)(inh_condition default))
              (((1782  1 ) z27_inh_slot1)
              (value nil)(inh_condition default))
              (((1883  1 ) z38_inh_slot1)
              (value nil)(inh_condition default))
              (((1984  1 ) z49_inh_slot1)
               (value z49_inh_v1)(inh_condition default)))))

((1200    z410)
 (id_slots ((offspring_no (value none))
            (level_no (value 5 ))
             (parents (value (1983 z39)))
            (children (value none))))
 (ind_slots ((nil (value nil))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1381  1 ) z13_inh_slot1)
              (value nil)(inh_condition default))
              (((1882  1 ) z28_ind_slot1)
              (value nil)(inh_condition default))
              (((1983  1 ) z39_inh_slot1)
              (value nil)(inh_condition default))
              (((1200  1 ) z410_inh_slot1)
               (value z410_inh_v1)(inh_condition default)))))

((1201    z411)
 (id_slots ((offspring_no (value none))
            (level_no (value 5 ))
             (parents (value (1983 z39)))
            (children (value none))))
 (ind_slots ((z411_ind_slot1 (value  z411_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1381  1 ) z13_inh_slot1)
               (value z411_inh_v2)(inh_condition union))
              (((1882  1 ) z28_ind_slot1)
              (value nil)(inh_condition default))
              (((1983  1 ) z39_inh_slot1)
              (value nil)(inh_condition default))
              (((1201  1 ) z411_inh_slot1)
               (value z411_inh_v3)(inh_condition default)))))

((1202    z412)
 (id_slots ((offspring_no (value none))
            (level_no (value 5 ))
               (parents (value ((1432 z311)(1583 z35))))
            (children (value none))))
 (ind_slots ((z412_ind_slot1 (value  z412_ind_v1))))
 (inh_slots ((((1381  1 ) z13_inh_slot1)
              (value nil)(inh_condition default))
              (((1982  1 ) z29_inh_slot1)
              (value nil)(inh_condition default))
              (((1432  1 ) z311_inh_slot1)
               (value z412_inh_v1)(inh_condition union))
              (((249  1 ) z1_inh_slot1)
              (value nil)(inh_condition default))
              (((1281  1 ) z12_inh_slot1)
               (value z412_inh_v2)(inh_condition override))
              (((1582  1 ) z25_inh_slot1)
               (value z412_inh_v3)(inh_condition union))
              (((1583  1 ) z35_inh_slot1)
               (value z412_inh_v4)(inh_condition union))
              (((1202  1 ) z412_inh_slot1)
               (value z412_inh_v5)(inh_condition default)))))

((1203    z413)
 (id_slots ((offspring_no (value none))
            (level_no (value 5 ))
             (parents (value (1432 z311)))
            (children (value none))))
 (ind_slots ((z413_ind_slot1 (value  z413_ind_v1))))
 (inh_slots ((((249   1 ) z1_inh_slot1)
               (value z413_inh_v1)(inh_condition override))
              (((1381  1 ) z13_inh_slot1)
               (value z413_inh_v2)(inh_condition override))
              (((1982  1 ) z29_inh_slot1)
               (value z413_inh_v3)(inh_condition union))
```

```
(((1432  1 ) z311_inh_slot1)
 (value z413_inh_v4)(inh_condition intersection))
(((1203  1 ) z413_inh_slot1)
 (value z413_inh_v5)(inh_condition default))
(((1203  2 ) z413_inh_slto2)
 (value z413_inh_v6)(inh_condition default)))))  )
```

In this appendix information on certain test runs is given. This information is produced at run time, at each execution, and can be used as a guide to the flow of execution.

Query query-object one

```
;(There are three options available for querying an object :)
;(1  - Asking for full definition of a Frame- giving the frame-name.)
;(2  - Asking for partial definition of a Frame- giving partial definition.)
;(3  - Asking for frame-name- giving its full or partial definition.)
;(choose one of the options)
1
;(type in frame-name :)
v47
;
;(Type_in file_name for your time delay)
;(Note : all the time-delay files have a prefix of main-disk:lisp:timedelay-
folder:filename)
;(eg : main-disk:lisp:timedelay-folder:time-delay1 where time-delay1 is a file name)
main-disk:lisp:timedelay-folder:td-45
;No_Mapping is required.
;(Propagation_Paths : (((1249  p1)(281  p12))((249  z1)(1181  z11)(1372  z21)(1373
z31))((1749  a1)(881  a13))((1749  a1)(681  a11)(682  a21))((1849  v1)(981  v13)(1800
v24)(1283  v36)(1384  v47))))
;(up_ward pointers : (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1)))
;(The_final_Paths : (((1249  p1)(281  p12))((249  z1)(1181  z11)(1372  z21)(1373
z31))((1749  a1)(881  a13))((1749  a1)(681  a11)(682  a21))((1849  v1)(981  v13)(1800
v24)(1283  v36)(1384  v47))))
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for p1)
;(A_packet added to p1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1249  p1) : (p1 (0  10 )(0  0 )(0  1.6 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for p12)
;(A_packet added to p12)
;(This_frame has a packet (1249  p1))
;(packet-size 295 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((p1_inh_slot1
(value p1_inh_v1))(p1_inh_slot2 (value p1_inh_v2)))))
;(THE-DISTANCE 1 )
```

```
;(THE-TRANSFER time 29.5 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (281 p12) : (p12 (1 10 )(0 29.5 )(1.6 10.2 )))
;(End_of_path : ((1249 p1)(281 p12)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for z1)
;(A_packet added to z1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (249 z1) : (z1 (0 27 )(0 0 )(0 .8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for z11)
;(A_packet added to z11)
;(This_frame has a packet (249 z1))
;(packet-size 269 )
;(the PACEKT (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1))((z1_inh_slot1
(value z1_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 53.8 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1181 z11) : (z11 (1 26 )(0 53.8 )(.8 3.7 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for z21)
;(A_packet added to z21)
;(This_frame has a packet (1181 z11))
;(packet-size 295 )
;(the PACEKT (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1))((z11_inh_slot1
(value z_inh_v1))(z1_inh_slot1 (value (z1_inh_v1))))))
;(THE-DISTANCE 4 )
;(THE-TRANSFER time 118. )
;(This_frame has a packet (281 p12))
;(packet-size 333 )
;(the PACEKT (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1))((p12_inh_slot1
(value p12_inh_v3))(p1_inh_slot2 (value (p1_inh_v2)))(p1_inh_slot1 (value
(p1_inh_v1 p12_inh_v1))))))
;(THE-DISTANCE 14 )
;(THE-TRANSFER time 466.2 )
;
```

```
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1372 z21) : (z21 (2 23 )(53.8 520. )(3.7 26.9 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for z31)
;(A_packet added to z31)
;(This_frame has a packet (1372 z21))
;(packet-size 424 )
;(the PACEKT (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1))((z21_inh_slot1
(value z21_inh_v4))(p12_inh_slot1 (value (p12_inh_v3 z21_inh_v3)))(p1_inh_slot2
(value (p1_inh_v2 z21_inh_v2)))(p1_inh_slot1 (value z21_inh_v1))(z11_inh_slot1
(value (z_inh_v1)))(z1_inh_slot1 (value (z1_inh_v1))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 84.8 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1373 z31) : (z31 (3 22 )(520. 604.8 )(26.9 72.4 )))
;(End_of_path : ((249 z1)(1181 z11)(1372 z21)(1373 z31)))
;---------------------------------
;---------------------------------
;Starting_Propagation .....
;(Packet_is made : (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1749 a1) : (a1 (0 3 )(0 0 )(0 2.8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a13)
;(A_packet added to a13)
;(This_frame has a packet (1749 a1))
;(packet-size 331 )
;(the PACEKT (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
a11_inh_slot1)((881 1 ) a13_inh_slot1)((981 1 ) v13_inh_slot1)((1800 1 )
v24_inh_slot1)((1283 1 ) v36_inh_slot1)((1384 1 ) v47_inh_slot1))((a1_inh_slot1
(value a1_v1))(a1_inh_slot2 (value a1_v2))(a1_inh_slot3 (value a1_v3))(a1_inh_slot4
(value a1_v4)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 66.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (881 a13) : (a13 (1 4 )(0 66.2 )(2.8 30.8 )))
;(End_of_path : ((1749 a1)(881 a13)))
;---------------------------------
;---------------------------------
;Starting_Propagation .....
;(Packet_is made : (nil (((1249 2 ) p1_inh_slot2)((281 1 ) p12_inh_slot1)((249 1 )
z1_inh_slot1)((1181 1 ) z11_inh_slot1)((1372 1 ) z21_inh_slot1)((1373 1 )
z31_inh_slot1)((1749 1 ) a1_inh_slot1)((1749 2 ) a1_inh_slot2)((681 1 )
```

```
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1749  a1) : (a1 (0  3 )(0  0 )(0  2.8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a11)
;(A_packet added to a11)
;(This_frame has a packet (1749  a1))
;(packet-size 331 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((a1_inh_slot1
(value a1_v1))(a1_inh_slot2 (value a1_v2))(a1_inh_slot3 (value a1_v3))(a1_inh_slot4
(value a1_v4)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 66.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (681  a11) : (a11 (1  2 )(0  66.2 )(2.8  30.8 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a21)
;(A_packet added to a21)
;(This_frame has a packet (681  a11))
;(packet-size 369 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((a11_inh_slot1
(value a11_inh_v4))(a1_inh_slot4 (value nil))(a1_inh_slot3 (value (a1_v3
a11_inh_v2)))(a1_inh_slot2 (value a11_inh_v1))(a1_inh_slot1 (value (a1_v1))))))
;(THE-DISTANCE 3 )
;(THE-TRANSFER time 110.7 )
;(This_frame has a packet (881  a13))
;(packet-size 369 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((a13_inh_slot1
(value a13_inh_v5))(a1_inh_slot4 (value (a1_v4)))(a1_inh_slot3 (value
nil))(a1_inh_slot2 (value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value a13_inh_v1)))))
;(THE-DISTANCE 5 )
;(THE-TRANSFER time 184.5 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (682  a21) : (a21 (2  0 )(66.2  250.7 )(30.8  112.4 )))
;(End_of_path : ((1749  a1)(681  a11)(682  a21)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
```

```
a11_inh_slot1)((881   1 ) a13_inh_slot1)((981   1 ) v13_inh_slot1)((1800   1 )
v24_inh_slot1)((1283   1 ) v36_inh_slot1)((1384   1 ) v47_inh_slot1)) nil))
;No_Mapping is required.
;(No_inheritance is rquired at v1)
;(Frame_ v1 Has_No parents)
;(A_packet added to v1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1849  v1) : (v1 (0  17 )(0  0 )(0  .8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for v13)
;(A_packet added to v13)
;(This_frame has a packet (1849  v1))
;(packet-size 269 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((v1_inh_slot1
(value v1_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 53.8 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (981  v13) : (v13 (1  18 )(0  53.8 )(.8  3.7 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for v24)
;(A_packet added to v24)
;(This_frame has a packet (981  v13))
;(packet-size 298 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((v13_inh_slot1
(value v13_inh_v2))(v1_inh_slot1 (value v13_inh_v1)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER time 29.8 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1800  v24) : (v24 (2  18 )(53.8  83.6 )(3.7  11.9 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for v36)
;(A_packet added to v36)
;(This_frame has a packet (1800  v24))
;(packet-size 336 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((v24_inh_slot1
(value v24_inh_v3))(v13_inh_slot1 (value (v13_inh_v2 v24_inh_v2)))(v1_inh_slot1
(value v24_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 67.2 )
;(This_frame has a packet (682  a21))
;(packet-size 500 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
```

```
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((a21_inh_slot1
(value a21_inh_v7))(a13_inh_slot1 (value (a13_inh_v5 a21_inh_v6)))(a11_inh_slot1
(value (a11_inh_v4 a21_inh_v5)))(a1_inh_slot4 (value a21_inh_v4))(a1_inh_slot3
(value (a1_v3 a11_inh_v2 a21_inh_v3)))(a1_inh_slot2 (value (a1_v2 a13_inh_v2
a11_inh_v1 a21_inh_v2)))(a1_inh_slot1 (value (a13_inh_v1 a1_v1))))))
;(THE-DISTANCE 20 )
;(THE-TRANSFER time 1000. )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1283  v36) : (v36 (3  19 )(83.6  1083.6 )(11.9  86.2 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for v47)
;(A_packet added to v47)
;(This_frame has a packet (1283  v36))
;(packet-size 616 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((v36_inh_slot1
(value v36_inh_v8))(v24_inh_slot1 (value (v24_inh_v3)))(v13_inh_slot1 (value
(v13_inh_v2 v24_inh_v2)))(a21_inh_slot1 (value v36_inh_v7))(a13_inh_slot1 (value
(a13_inh_v5 a21_inh_v6 v36_inh_v6)))(a11_inh_slot1 (value nil))(a1_inh_slot4 (value
v36_inh_v4))(a1_inh_slot3 (value v36_inh_v3))(a1_inh_slot2 (value (a1_v2
a13_inh_v2 a11_inh_v1 a21_inh_v2 v36_inh_v2)))(a1_inh_slot1 (value (a13_inh_v1
a1_v1 v36_inh_v1)))(v1_inh_slot1 (value v24_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 123.2 )
;(This_frame has a packet (1373  z31))
;(packet-size 472 )
;(the PACEKT (nil (((1249  2 ) p1_inh_slot2)((281  1 ) p12_inh_slot1)((249  1 )
z1_inh_slot1)((1181  1 ) z11_inh_slot1)((1372  1 ) z21_inh_slot1)((1373  1 )
z31_inh_slot1)((1749  1 ) a1_inh_slot1)((1749  2 ) a1_inh_slot2)((681  1 )
a11_inh_slot1)((881  1 ) a13_inh_slot1)((981  1 ) v13_inh_slot1)((1800  1 )
v24_inh_slot1)((1283  1 ) v36_inh_slot1)((1384  1 ) v47_inh_slot1))((z31_inh_slot1
(value z31_inh_v3))(z21_inh_slot1 (value (z21_inh_v4)))(z11_inh_slot1 (value
(z_inh_v1)))(z1_inh_slot1 (value (z1_inh_v1)))(p12_inh_slot1 (value (p12_inh_v3
z21_inh_v3 z31_inh_v2)))(p1_inh_slot2 (value (p1_inh_v2 z21_inh_v2
z31_inh_v1)))(p1_inh_slot1 (value z21_inh_v1)))))
;(THE-DISTANCE 3 )
;(THE-TRANSFER time 141.6 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-34)
;(Time_delay for (1384  v47) : (v47 (4  20 )(1083.6  1225.2 )(86.2  284.9 )))
;(End_of_path : ((1849  v1)(981  v13)(1800  v24)(1283  v36)(1384  v47)))
;_____________________________
;(v47 ((v47_inh_slot1 (value v47_inh_v11))(v36_inh_slot1 (value
(v36_inh_v8)))(v24_inh_slot1 (value (v24_inh_v3)))(v13_inh_slot1 (value (v13_inh_v2
v24_inh_v2)))(a13_inh_slot1 (value (a13_inh_v5 a21_inh_v6 v36_inh_v6
v47_inh_v10)))(a11_inh_slot1 (value (v47_inh_v9)))(a1_inh_slot2 (value (a1_v2
a13_inh_v2 a11_inh_v1 a21_inh_v2 v36_inh_v2 v47_inh_v8)))(a1_inh_slot1 (value
(a13_inh_v1 a1_v1 v36_inh_v1 v47_inh_v7)))(z31_inh_slot1 (value (z31_inh_v3
v47_inh_v6)))(z21_inh_slot1 (value (z21_inh_v4 v47_inh_v5)))(z11_inh_slot1 (value
(z_inh_v1 v47_inh_v4)))(z1_inh_slot1 (value v47_inh_v3))(p12_inh_slot1 (value
(p12_inh_v3 z21_inh_v3 z31_inh_v2 v47_inh_v2)))(p1_inh_slot2 (value (p1_inh_v2
z21_inh_v2 z31_inh_v1 v47_inh_v1)))(a21_inh_slot1 (value
v36_inh_v7))(a1_inh_slot4 (value v36_inh_v4))(a1_inh_slot3 (value
v36_inh_v3))(v1_inh_slot1 (value v24_inh_v1))(p1_inh_slot1 (value z21_inh_v1))))
```

```
;(v47 ((v47_inh_slot1 (value v47_inh_v11))(v36_inh_slot1 (value
(v36_inh_v8)))(v24_inh_slot1 (value (v24_inh_v3)))(v13_inh_slot1 (value (v13_inh_v2
v24_inh_v2)))(a13_inh_slot1 (value (a13_inh_v5 a21_inh_v6 v36_inh_v6
v47_inh_v10)))(a11_inh_slot1 (value (v47_inh_v9)))(a1_inh_slot2 (value (a1_v2
a13_inh_v2 a11_inh_v1 a21_inh_v2 v36_inh_v2 v47_inh_v8)))(a1_inh_slot1 (value
(a13_inh_v1 a1_v1 v36_inh_v1 v47_inh_v7)))(z31_inh_slot1 (value (z31_inh_v3
v47_inh_v6)))(z21_inh_slot1 (value (z21_inh_v4 v47_inh_v5)))(z11_inh_slot1 (value
(z_inh_v1 v47_inh_v4)))(z1_inh_slot1 (value v47_inh_v3))(p12_inh_slot1 (value
(p12_inh_v3 z21_inh_v3 z31_inh_v2 v47_inh_v2)))(p1_inh_slot2 (value (p1_inh_v2
z21_inh_v2 z31_inh_v1 v47_inh_v1)))(a21_inh_slot1 (value
v36_inh_v7))(a1_inh_slot4 (value v36_inh_v4))(a1_inh_slot3 (value
v36_inh_v3))(v1_inh_slot1 (value v24_inh_v1))(p1_inh_slot1 (value z21_inh_v1))))
```

Query query-object two

```
;(There are three options available for querying an object :)
;(1  - Asking for full definition of a Frame- giving the frame-name.)
;(2  - Asking for partial definition of a Frame- giving partial definition.)
;(3  - Asking for frame-name- giving its full or partial definition.)
;(choose one of the options)
2
;(type in the object_name :)
a32
;(Creating a query_frame for the object a32)
;(You are processing a32)
;(press any key)

;(type in slot_name 1 )
;(if no slots cr)
a1_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)

;(type in slot_name 2 )
;(if no slots cr)
a1_inh_slot2
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)

;(type in slot_name 3 )
;(if no slots cr)
a11_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)

;(type in slot_name 4 )
;(if no slots cr)
a13_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)
```

```
;(type in slot_name 5 )
;(if no slots cr)
v1_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)
p1_inh_slot1
;(type in slot_name 6 )
;(if no slots cr)

;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)

;(type in slot_name 7 )
;(if no slots cr)
p1_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop processing a32 otherwise cr)
end
;
;(Type_in file_name for your time delay)
;(Note : all the time-delay files have a prefix of main-disk:lisp:timedelay-
folder:filename)
;(eg : main-disk:lisp:timedelay-folder:time-delay1 where time-delay1 is a file name)
main-disk:lisp:timedelay-folder:td-55
;No_Mapping is required.
;(Propagation_Paths : (((1249 p1)(181 p11)(182 p21))((1849 v1))((1749 a1)(781
a12)(882 a23))((1749 a1)(881 a13))((1749 a1)(681 a11)(682 a21)(783 a32))))
;(up_ward pointers : (((1249 1 ) p1_inh_slot1)((1849 v1) v1_inh_slot1)((881 1 )
a13_inh_slot1)((681 1 ) a11_inh_slot1)((1749 2 ) a1_inh_slot2)((1749 1 )
a1_inh_slot1)))
;(The_final_Paths : (((1249 p1)(181 p11)(182 p21))((1849 v1))((1749 a1)(781
a12)(882 a23))((1749 a1)(881 a13))((1749 a1)(681 a11)(682 a21)(783 a32))))
;---------------------------------
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249 1 ) p1_inh_slot1)((1849 v1)
v1_inh_slot1)((881 1 ) a13_inh_slot1)((681 1 ) a11_inh_slot1)((1749 2 )
a1_inh_slot2)((1749 1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for p1)
;(A_packet added to p1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (1249 p1) : (p1 (0 10 )(0 0 )(0 .8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for p11)
;(A_packet added to p11)
;(This_frame has a packet (1249 p1))
;(packet-size 238 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249 1 ) p1_inh_slot1)((1849 v1)
```

```
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1))((p1_inh_slot1 (value p1_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 47.6 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (181 p11) : (p11 (1 9 )(0 47.6 )(.8 3.1 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for p21)
;(A_packet added to p21)
;(This_frame has a packet (181 p11))
;(packet-size 248 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1))((p1_inh_slot1 (value (p1_inh_v1
p11_inh_v1))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 49.6 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (182 p21) : (p21 (2 8 )(47.6 97.2 )(3.1 6.9 )))
;(End_of_path : ((1249 p1)(181 p11)(182 p21)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for v1)
;(A_packet added to v1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (1849 v1) : (v1 (0 17 )(0 0 )(0 .8 )))
;(End_of_path : ((1849 v1)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (1749 a1) : (a1 (0 3 )(0 0 )(0 1.4 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a12)
;(A_packet added to a12)
;(This_frame has a packet (1749 a1))
;(packet-size 256 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
```

```
nil))(p1_inh_slot1 (value nil)))(((1249  1 ) p1_inh_slot1)((1849  v1)
v1_inh_slot1)((881  1 ) a13_inh_slot1)((681  1 ) a11_inh_slot1)((1749  2 )
a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot2 (value a1_v2))(a1_inh_slot1
(value a1_v1)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER time 25.6 )
;(This_frame has a packet (1849  v1))
;(packet-size 238 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249  1 ) p1_inh_slot1)((1849  v1)
v1_inh_slot1)((881  1 ) a13_inh_slot1)((681  1 ) a11_inh_slot1)((1749  2 )
a1_inh_slot2)((1749  1 ) a1_inh_slot1))((v1_inh_slot1 (value v1_inh_v1)))))
;(THE-DISTANCE 15 )
;(THE-TRANSFER time 357. )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (781  a12) : (a12 (1  3 )(0  357. )(1.4  9.3 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a23)
;(A_packet added to a23)
;(This_frame has a packet (781  a12))
;(packet-size 302 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249  1 ) p1_inh_slot1)((1849  v1)
v1_inh_slot1)((881  1 ) a13_inh_slot1)((681  1 ) a11_inh_slot1)((1749  2 )
a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot2 (value (a1_v2
a12_inh_v2)))(a1_inh_slot1 (value a1_v1))(v1_inh_slot1 (value (v1_inh_v1
a12_inh_v5))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 60.4 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (882  a23) : (a23 (2  2 )(357.  417.4 )(9.3  23.7 )))
;(End_of_path : ((1749  a1)(781  a12)(882  a23)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249  1 ) p1_inh_slot1)((1849  v1)
v1_inh_slot1)((881  1 ) a13_inh_slot1)((681  1 ) a11_inh_slot1)((1749  2 )
a1_inh_slot2)((1749  1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (1749  a1) : (a1 (0  3 )(0  0 )(0  1.4 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a13)
;(A_packet added to a13)
;(This_frame has a packet (1749  a1))
;(packet-size 256 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249  1 ) p1_inh_slot1)((1849  v1)
v1_inh_slot1)((881  1 ) a13_inh_slot1)((681  1 ) a11_inh_slot1)((1749  2 )
```

```
a1_inh_slot2)((1749    1  )  a1_inh_slot1))((a1_inh_slot2  (value  a1_v2))(a1_inh_slot1
(value  a1_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 51.2 )
;
;(Writing-time  delay  to  main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (881  a13) : (a13 (1  4 )(0  51.2 )(1.4  8.4 )))
;(End_of_path : ((1749  a1)(881  a13)))
;_____________________________
;_____________________________
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time  delay  to  main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (1749  a1) : (a1 (0  3 )(0  0 )(0  1.4 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a11)
;(A_packet added to a11)
;(This_frame has a packet (1749  a1))
;(packet-size 256 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1))((a1_inh_slot2  (value  a1_v2))(a1_inh_slot1
(value  a1_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 51.2 )
;
;(Writing-time  delay  to  main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (681  a11) : (a11 (1  2 )(0  51.2 )(1.4  8.4 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a21)
;(A_packet added to a21)
;(This_frame has a packet (681  a11))
;(packet-size 289 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1))((a1_inh_slot2 (value
a11_inh_v1))(a1_inh_slot1  (value  a1_v1))(a11_inh_slot1  (value  a11_inh_v4)))))
;(THE-DISTANCE 3 )
;(THE-TRANSFER time 86.7 )
;(This_frame has a packet (881  a13))
;(packet-size 299 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249   1 ) p1_inh_slot1)((1849   v1)
v1_inh_slot1)((881   1 ) a13_inh_slot1)((681   1 ) a11_inh_slot1)((1749   2 )
a1_inh_slot2)((1749   1 ) a1_inh_slot1))((a1_inh_slot2 (value (a1_v2
a13_inh_v2)))(a1_inh_slot1  (value  a13_inh_v1))(a13_inh_slot1  (value  a13_inh_v5)))))
;(THE-DISTANCE 5 )
```

```
;(THE-TRANSFER time 149.5 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (682 a21) : (a21 (2 0 )(51.2 200.7 )(8.4 27.4 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a32)
;(A_packet added to a32)
;(This_frame has a packet (682 a21))
;(packet-size 406 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249 1 ) p1_inh_slot1)((1849 v1)
v1_inh_slot1)((881 1 ) a13_inh_slot1)((681 1 ) a11_inh_slot1)((1749 2 )
a1_inh_slot2)((1749 1 ) a1_inh_slot1))((a1_inh_slot2 (value (a11_inh_v1
a21_inh_v2)))(a1_inh_slot1 (value a1_v1))(a11_inh_slot1 (value (a11_inh_v4
a21_inh_v5)))(a1_inh_slot2 (value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value
a13_inh_v1))(a13_inh_slot1 (value (a13_inh_v5 a21_inh_v6))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 81.2 )
;(This_frame has a packet (882 a23))
;(packet-size 322 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249 1 ) p1_inh_slot1)((1849 v1)
v1_inh_slot1)((881 1 ) a13_inh_slot1)((681 1 ) a11_inh_slot1)((1749 2 )
a1_inh_slot2)((1749 1 ) a1_inh_slot1))((a1_inh_slot2 (value (a1_v2 a12_inh_v2
a23_inh_v2)))(a1_inh_slot1 (value a1_v1))(v1_inh_slot1 (value (v1_inh_v1 a12_inh_v5
a23_inh_v6))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 64.4 )
;(This_frame has a packet (182 p21))
;(packet-size 258 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a11_inh_slot1
(value nil))(a13_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(nil (value
nil))(p1_inh_slot1 (value nil)))(((1249 1 ) p1_inh_slot1)((1849 v1)
v1_inh_slot1)((881 1 ) a13_inh_slot1)((681 1 ) a11_inh_slot1)((1749 2 )
a1_inh_slot2)((1749 1 ) a1_inh_slot1))((p1_inh_slot1 (value (p1_inh_v1 p11_inh_v1
p21_inh_v1))))))
;(THE-DISTANCE 8 )
;(THE-TRANSFER time 206.4 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-55)
;(Time_delay for (783 a32) : (a32 (3 1 )(200.7 407.1 )(27.4 66.5 )))
;(End_of_path : ((1749 a1)(681 a11)(682 a21)(783 a32)))
;_____________________________
;(a32 ((a1_inh_slot2 (value (a11_inh_v1 a21_inh_v2 a32_inh_v2)))(a1_inh_slot1 (value
(a1_v1 a32_inh_v1)))(a11_inh_slot1 (value (a11_inh_v4 a21_inh_v5
a32_inh_v5)))(a1_inh_slot2 (value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value
a13_inh_v1))(a13_inh_slot1 (value nil))(a1_inh_slot2 (value (a1_v2 a12_inh_v2
a23_inh_v2)))(a1_inh_slot1 (value a1_v1))(v1_inh_slot1 (value (v1_inh_v1 a12_inh_v5
a23_inh_v6)))(p1_inh_slot1 (value (p1_inh_v1 p11_inh_v1 p21_inh_v1
a32_inh_v11)))))
;(a32 ((a1_inh_slot2 (value (a11_inh_v1 a21_inh_v2 a32_inh_v2)))(a1_inh_slot1 (value
(a1_v1 a32_inh_v1)))(a11_inh_slot1 (value (a11_inh_v4 a21_inh_v5
a32_inh_v5)))(a1_inh_slot2 (value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value
a13_inh_v1))(a13_inh_slot1 (value nil))(a1_inh_slot2 (value (a1_v2 a12_inh_v2
a23_inh_v2)))(a1_inh_slot1 (value a1_v1))(v1_inh_slot1 (value (v1_inh_v1 a12_inh_v5
a23_inh_v6)))(p1_inh_slot1 (value (p1_inh_v1 p11_inh_v1 p21_inh_v1
a32_inh_v11)))))
```

Query query-object three

```
;(There are three options available for querying an object :)
;(1  - Asking for full definition of a Frame- giving the frame-name.)
;(2  - Asking for partial definition of a Frame- giving partial definition.)
;(3  - Asking for frame-name- giving its full or partial definition.)
;(choose one of the options)
3
;(type in hierarchy_names :)
(A-hir V-hir P-hir)
;(Creating a query_frame for the object ?frame)
;(press any key)

;(type in slot_name 1 )
;(if no slots cr)
a1_inh_slot1
;(any value for this slot)
;(if no value cr)
(a1_v1  a32_inh_v1)
;(type end to stop- otherwise cr)

;(type in slot_name 2 )
;(if no slots cr)
a1_inh_slot2
;(any value for this slot)
;(if no value cr)
(a11_inh_v1  a21_inh_v2  a32_inh_v2)
;(type end to stop- otherwise cr)

;(type in slot_name 3 )
;(if no slots cr)
a1_inh_slot3
;(any value for this slot)
;(if no value cr)
(a1_v3  a11_inh_v2  a21_inh_v3  a32_inh_v3)
;(type end to stop- otherwise cr)

;(type in slot_name 4 )
;(if no slots cr)
v1_inh_slot1
;(any value for this slot)
;(if no value cr)
(v1_inh_v1  a12_inh_v5  a23_inh_v6)
;(type end to stop- otherwise cr))

;(type in slot_name 5 )
;(if no slots cr)
p1_inh_slot1
;(any value for this slot)
;(if no value cr)
(p1_inh_v1  p11_inh_v1  p21_inh_v1  a32_inh_v11)
;(type end to stop- otherwise cr))

;(type in slot_name 6 )
;(if no slots cr)
p21_inh_slot1
;(any value for this slot)
;(if no value cr)
(p21_inh_v4  a32_inh_v14)
;(type end to stop- otherwise cr))
```

```
;(type in slot_name 7 )
;(if no slots cr)
a32_inh_slot1
;(any value for this slot)
;(if no value cr)
a32_inh_v15
;(type end to stop- otherwise cr))
end
;
;(Type_in file_name for your time delay)
;(Note : all the time-delay files have a prefix of main-disk:lisp:timedelay-
folder:filename)
;(eg : main-disk:lisp:timedelay-folder:time-delay1 where time-delay1 is a file name)
main-disk:lisp:timedelay-folder:td-60
;No_Mapping is required.
;_______________________________
;Starting_Propagation .....
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3
(value nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for p1)
;(A_packet added to p1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (1249  p1) : (p1 (0  10 )(0  0 )(0  .8 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for p11)
;(A_packet added to p11)
;(This_frame has a packet (1249  p1))
;(packet-size 266 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((p1_inh_slot1 (value
p1_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 53.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (181  p11) : (p11 (1  9 )(0  53.2 )(.8  3.1 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for p21)
;(A_packet added to p21)
;(This_frame has a packet (181  p11))
;(packet-size 276 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((p1_inh_slot1 (value
(p1_inh_v1  p11_inh_v1))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 55.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
```

```
;(Time_delay for (182  p21) : (p21 (2  8 )(53.2  108.4 )(3.1  7.4 )))
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3
(value nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for v1)
;(A_packet added to v1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (1849  v1) : (v1 (0  17 )(0  0 )(0  .8 )))
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3
(value nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (1749  a1) : (a1 (0  3 )(0  0 )(0  2.1 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a12)
;(A_packet added to a12)
;(This_frame has a packet (1749  a1))
;(packet-size 306 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
a1_v3))(a1_inh_slot2 (value a1_v2))(a1_inh_slot1 (value a1_v1)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER time 30.6 )
;(This_frame has a packet (1849  v1))
;(packet-size 266 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((v1_inh_slot1 (value
v1_inh_v1)))))
;(THE-DISTANCE 15 )
;(THE-TRANSFER time 399. )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (781  a12) : (a12 (1  3 )(0  399. )(2.1  15.3 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a23)
;(A_packet added to a23)
;(This_frame has a packet (781  a12))
;(packet-size 362 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
```

```
(a1_v3 a12_inh_v3)))(a1_inh_slot2 (value (a1_v2 a12_inh_v2)))(a1_inh_slot1 (value
a1_v1))(v1_inh_slot1 (value (v1_inh_v1 a12_inh_v5))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 72.4 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (882 a23) : (a23 (2 2 )(399. 471.4 )(15.3 39.6 )))
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3
(value nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783 1 ) a32_inh_slot1)((182 1 )
p21_inh_slot1)((1249 1 ) p1_inh_slot1)((1849 1 ) v1_inh_slot1)((1749 3 )
a1_inh_slot3)((1749 2 ) a1_inh_slot2)((1749 1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (1749 a1) : (a1 (0 3 )(0 0 )(0 2.1 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a13)
;(A_packet added to a13)
;(This_frame has a packet (1749 a1))
;(packet-size 306 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783 1 ) a32_inh_slot1)((182 1 )
p21_inh_slot1)((1249 1 ) p1_inh_slot1)((1849 1 ) v1_inh_slot1)((1749 3 )
a1_inh_slot3)((1749 2 ) a1_inh_slot2)((1749 1 ) a1_inh_slot1))((a1_inh_slot3 (value
a1_v3))(a1_inh_slot2 (value a1_v2))(a1_inh_slot1 (value a1_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 61.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (881 a13) : (a13 (1 4 )(0 61.2 )(2.1 12.9 )))
;(Packet_is made : (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3
(value nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783 1 ) a32_inh_slot1)((182 1 )
p21_inh_slot1)((1249 1 ) p1_inh_slot1)((1849 1 ) v1_inh_slot1)((1749 3 )
a1_inh_slot3)((1749 2 ) a1_inh_slot2)((1749 1 ) a1_inh_slot1)) nil))
;No_Mapping is required.
;(No_multiparentage only inheritance for a1)
;(A_packet added to a1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (1749 a1) : (a1 (0 3 )(0 0 )(0 2.1 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a11)
;(A_packet added to a11)
;(This_frame has a packet (1749 a1))
;(packet-size 306 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783 1 ) a32_inh_slot1)((182 1 )
p21_inh_slot1)((1249 1 ) p1_inh_slot1)((1849 1 ) v1_inh_slot1)((1749 3 )
a1_inh_slot3)((1749 2 ) a1_inh_slot2)((1749 1 ) a1_inh_slot1))((a1_inh_slot3 (value
a1_v3))(a1_inh_slot2 (value a1_v2))(a1_inh_slot1 (value a1_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 61.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
```

```
;(Time_delay for (681  a11) : (a11 (1  2 )(0  61.2 )(2.1  12.9 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a21)
;(A_packet added to a21)
;(This_frame has a packet (681  a11))
;(packet-size 321 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
(a1_v3 a11_inh_v2)))(a1_inh_slot2 (value a11_inh_v1))(a1_inh_slot1 (value a1_v1)))))
;(THE-DISTANCE 3 )
;(THE-TRANSFER time 96.3 )
;(This_frame has a packet (881  a13))
;(packet-size 316 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
nil))(a1_inh_slot2 (value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value a13_inh_v1)))))
;(THE-DISTANCE 5 )
;(THE-TRANSFER time 158. )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (682  a21) : (a21 (2  0 )(61.2  219.2 )(12.9  32.4 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for a32)
;(A_packet added to a32)
;(This_frame has a packet (682  a21))
;(packet-size 417 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
(a1_v3 a11_inh_v2 a21_inh_v3)))(a1_inh_slot2 (value (a11_inh_v1
a21_inh_v2)))(a1_inh_slot1 (value a1_v1))(a1_inh_slot3 (value nil))(a1_inh_slot2
(value (a1_v2 a13_inh_v2)))(a1_inh_slot1 (value a13_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 83.4 )
;(This_frame has a packet (882  a23))
;(packet-size 392 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((a1_inh_slot3 (value
(a1_v3 a12_inh_v3 a23_inh_v3)))(a1_inh_slot2 (value (a1_v2 a12_inh_v2
a23_inh_v2)))(a1_inh_slot1 (value a1_v1))(v1_inh_slot1 (value (v1_inh_v1 a12_inh_v5
a23_inh_v6))))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 78.4 )
;(This_frame has a packet (182  p21))
;(packet-size 314 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783  1 ) a32_inh_slot1)((182  1 )
p21_inh_slot1)((1249  1 ) p1_inh_slot1)((1849  1 ) v1_inh_slot1)((1749  3 )
```

```
a1_inh_slot3)((1749  2 ) a1_inh_slot2)((1749  1 ) a1_inh_slot1))((p1_inh_slot1 (value
(p1_inh_v1 p11_inh_v1 p21_inh_v1)))(p21_inh_slot1 (value p21_inh_v4)))))
;(THE-DISTANCE 8 )
;(THE-TRANSFER time 251.2 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (783 a32) : (a32 (3 1 )(219.2 470.4 )(32.4 78.2 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for a41)
;(A_packet added to a41)
;(This_frame has a packet (783 a32))
;(packet-size 724 )
;(the PACEKT (((a1_inh_slot1 (value nil))(a1_inh_slot2 (value nil))(a1_inh_slot3 (value
nil))(v1_inh_slot1 (value nil))(p1_inh_slot1 (value nil))(p21_inh_slot1 (value
nil))(a32_inh_slot1 (value nil)))(((783 1 ) a32_inh_slot1)((182 1 )
p21_inh_slot1)((1249 1 ) p1_inh_slot1)((1849 1 ) v1_inh_slot1)((1749 3 )
a1_inh_slot3)((1749 2 ) a1_inh_slot2)((1749 1 ) a1_inh_slot1))((a1_inh_slot3 (value
(a1_v3 a11_inh_v2 a21_inh_v3 a32_inh_v3)))(a1_inh_slot2 (value (a11_inh_v1
a21_inh_v2 a32_inh_v2)))(a1_inh_slot1 (value (a1_v1 a32_inh_v1)))(v1_inh_slot1
(value (v1_inh_v1 a12_inh_v5 a23_inh_v6)))(p1_inh_slot1 (value (p1_inh_v1
p11_inh_v1 p21_inh_v1 a32_inh_v11)))(p21_inh_slot1 (value (p21_inh_v4
a32_inh_v14)))(a32_inh_slot1 (value a32_inh_v15)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER time 144.8 )
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-70)
;(Time_delay for (684 a41) : (a41 (4 0 )(470.4 615.2 )(78.2 144.1 )))
;--------------------------------
;(a41)
```

Query Domain_2

```
;(There are two types of domain-related queries :)
;
;(1  - Asking for subclass relationships.)
;
;(2  - Asking for some shared characteristics between two or more frames.)
;
;
;(choose one of the numbers)
2
;(type in the objects with their slots and their values)
;(Note : in the absence of any one of these items a system variable will be placed in its
position)
;(press any key)

;(type in frame-name 1 )
;(if no frame-name cr)
z23
;(type in frame-name 2 )
;(if no frame-name cr)
z34
;(type in frame-name 3 )
;(if no frame-name cr)

;(type in slot_name 1 )
;(if no slots cr)
z1_inh_slot1
;(any value for this slot)
```

```
;(if no value cr)

;(type end to stop- otherwise cr)

;(type in slot_name 2 )
;(if no slots cr)
v1_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop- otherwise cr)

;(type in slot_name 3 )
;(if no slots cr)
v13_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop- otherwise cr)

;(type in slot_name 4 )
;(if no slots cr)
z23_inh_slot1
;(any value for this slot)
;(if no value cr)

;(type end to stop- otherwise cr)
end
;(Type_in file_name for your time delay)
;(Note : all the time-delay files have a prefix of main-disk:lisp:timedelay-folder:)
main-disk:lisp:timedelay-folder:td-86
;(Type_in file_name for prompts)
main-disk:lisp:timedelay-folder:td-p
;No_Mapping is required.
;(Propagation_Paths : (((1849  v1)(981  v13))((249  z1)(1281  z12)(1382  z23)(605
z34))))
;(up_ward pointers : (((v1_inh_slot1 (value nil)))(((981  1 ) v13_inh_slot1)((249  1 )
z1_inh_slot1)((1382  1 ) z23_inh_slot1))))
;Starting_Propagation .....
;(A_Packet_is made.)
;No_Mapping is required.
;(No_inheritance is rquired at v1)
;(Frame_ v1 Has_No parents)
;(A_packet added to v1)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1849  v1) : (v1 (0  17 )(7.  7. )(13.6  14.4 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for v13)
;(A_packet added to v13)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (981  v13) : (v13 (1  18 )(7.  36.4 )(14.4  17.3 )))
;(End_of_path : ((1849  v1)(981  v13)))
;Starting_Propagation .....
;(A_Packet_is made.)
;No_Mapping is required.
;(No_multiparentage only inheritance for z1)
;(A_packet added to z1)
;
```

```
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (249 z1) : (z1 (0 27 )(7. 7. )(13.6 14.4 )))
;No_Mapping is required.
;(No_multiparentage only inheritance for z12)
;(A_packet added to z12)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1281 z12) : (z12 (1 27 )(7. 21.7 )(14.4 16.7 )))
;No_Mapping is required.
;(Both_inheritance and multiparentage for z23)
;(A_packet added to z23)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1382 z23) : (z23 (2 25 )(21.7 162.5 )(16.7 26.3 )))
;(A_packet added to z33)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (605 z34) : (z34 (3 25 )(162.5 187.5 )(26.3 38.1 )))
;(A_packet added to z42)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1605 z33) : (z33 (3 24 )(162.5 212.5 )(26.3 38.1 )))
;(A_packet added to z45)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1584 z45) : (z45 (4 25 )(212.5 264.5 )(38.1 53.2 )))
;(A_packet added to z34)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (374 z42) : (z42 (4 22 )(187.5 291.5 )(38.1 50.2 )))
;(A_packet added to z34)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (605 z34) : (z34 (3 25 )(162.5 187.5 )(26.3 38.1 )))
;(A_packet added to z43)
;
;(Writing-time delay to main-disk:lisp:timedelay-folder:td-86)
;(Time_delay for (1606 z43) : (z43 (4 23 )(187.5 265.5 )(38.1 53.5 )))
;(Packet-for (1849 v1) : (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value
nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981 1 )
v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 ) z23_inh_slot1))((v1_inh_slot1 (value
v1_inh_v1)))))
;(This_frame-has-a-packet (1849 v1))
;(packet-size 147 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((v1_inh_slot1 (value v1_inh_v1)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER-TIME 29.4 )
;(Packet-for (981 v13) : (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value
nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981 1 )
v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 ) z23_inh_slot1))((v1_inh_slot1 (value
v13_inh_v1))(v13_inh_slot1 (value v13_inh_v2)))))
;(Packet-for (249 z1) : (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value
nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981 1 )
v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value
z1_inh_v1)))))
;(This_frame-has-a-packet (249 z1))
;(packet-size 147 )
```

```
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value z1_inh_v1)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER-TIME 14.7 )
;(Packet-for (1281 z12) : (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value
nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981 1 )
v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value
z1_inh_v1)))))
;(This_frame-has-a-packet (1281 z12))
;(packet-size 147 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value z1_inh_v1)))))
;(THE-DISTANCE 3 )
;(THE-TRANSFER-TIME 44.1 )
;(This_frame-has-a-packet (981 v13))
;(packet-size 176 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((v1_inh_slot1 (value
v13_inh_v1))(v13_inh_slot1 (value v13_inh_v2)))))
;(THE-DISTANCE 8 )
;(THE-TRANSFER-TIME 140.8 )
;(Packet-for (1382 z23) : (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value
nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981 1 )
v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value
(z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value
(v13_inh_v2 z23_inh_v4)))(z23_inh_slot1 (value z23_inh_v5)))))
;(A_packet is added to (1605 z33) The_Packet (((z1_inh_slot1 (value
nil))(v1_inh_slot1 (value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value
nil)))(((981 1 ) v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 )
z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value
z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2 z23_inh_v4
z33_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;(This_frame-has-a-packet (1382 z23))
;(packet-size 250 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4)))(z23_inh_slot1 (value z23_inh_v5)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER-TIME 50. )
;(A_packet is added to (1584 z45) The_Packet (((z1_inh_slot1 (value
nil))(v1_inh_slot1 (value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value
nil)))(((981 1 ) v13_inh_slot1)((249 1 ) z1_inh_slot1)((1382 1 )
z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value
z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2 z23_inh_v4 z33_inh_v1
z45_inh_v3)))(z23_inh_slot1 (value z45_inh_v4)))))
;(This_frame-has-a-packet (1605 z33))
;(packet-size 260 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981 1 ) v13_inh_slot1)((249 1 )
z1_inh_slot1)((1382 1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4 z33_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;(THE-DISTANCE 2 )
;(THE-TRANSFER-TIME 52. )
```

```
;(A_packet is added to (605  z34) The_Packet (((z1_inh_slot1 (value nil))(v1_inh_slot1
(value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981  1 )
v13_inh_slot1)((249  1 ) z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value
(z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value
(v13_inh_v2 z23_inh_v4 z34_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;(This_frame-has-a-packet (1382  z23))
;(packet-size 250 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981  1 ) v13_inh_slot1)((249  1 )
z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4)))(z23_inh_slot1 (value z23_inh_v5)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER-TIME 25. )
;(A_packet is added to (374  z42) The_Packet (((z1_inh_slot1 (value nil))(v1_inh_slot1
(value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981  1 )
v13_inh_slot1)((249  1 ) z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value
(z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value
(v13_inh_v2 z23_inh_v4 z34_inh_v1 z42_inh_v1)))(z23_inh_slot1 (value
z23_inh_v5)))))
;(This_frame-has-a-packet (605  z34))
;(packet-size 260 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981  1 ) v13_inh_slot1)((249  1 )
z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4 z34_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;(THE-DISTANCE 4 )
;(THE-TRANSFER-TIME 104. )
;(A_packet is added to (605  z34) The_Packet (((z1_inh_slot1 (value nil))(v1_inh_slot1
(value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value nil)))(((981  1 )
v13_inh_slot1)((249  1 ) z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value
(z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value
(v13_inh_v2 z23_inh_v4 z34_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;(This_frame-has-a-packet (1382  z23))
;(packet-size 250 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981  1 ) v13_inh_slot1)((249  1 )
z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4)))(z23_inh_slot1 (value z23_inh_v5)))))
;(THE-DISTANCE 1 )
;(THE-TRANSFER-TIME 25. )
;(A_packet is added to (1606  z43) The_Packet (((z1_inh_slot1 (value
nil))(v1_inh_slot1 (value nil))(v13_inh_slot1 (value nil))(z23_inh_slot1 (value
nil)))(((981  1 ) v13_inh_slot1)((249  1 ) z1_inh_slot1)((1382  1 )
z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value
z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2 z23_inh_v4 z34_inh_v1
z43_inh_v1)))(z23_inh_slot1 (value (z23_inh_v5 z43_inh_v2))))))
;(This_frame-has-a-packet (605  z34))
;(packet-size 260 )
;(THE-PACEKT (((z1_inh_slot1 (value nil))(v1_inh_slot1 (value nil))(v13_inh_slot1
(value nil))(z23_inh_slot1 (value nil)))(((981  1 ) v13_inh_slot1)((249  1 )
z1_inh_slot1)((1382  1 ) z23_inh_slot1))((z1_inh_slot1 (value (z1_inh_v1
z23_inh_v1)))(v1_inh_slot1 (value z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2
z23_inh_v4 z34_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)))))
;
;
```

```
;((z23 z34)) (z1_inh_slot1 (value (z1_inh_v1 z23_inh_v1)))(v1_inh_slot1 (value
z23_inh_v3))(v13_inh_slot1 (value (v13_inh_v2 z23_inh_v4
z34_inh_v1)))(z23_inh_slot1 (value z23_inh_v5)) )
;
```

APPENDIX D : GRAPHS

In this appendix, the graphical results of some of the test runs on frame-related conjunctive queries of type query-object one, query-object two, query-object three and domain-related conjunctive queries are presented. For each query made, five different items are produced :

1 - Query-frame
2 - Time-list
3 - Run-time graph
4 - Graph number 1/2
5 - Graph number 5

D1.0 QUERIES OF TYPE QUERY-OBJECT NUMBER ONE

In this type of query the user provides the frame-name, and the system, in return, will find its full definition. Therefore, there is no query-frame, and only 4 items are presented for each query.

1.1 QUERY OBJECT : a12

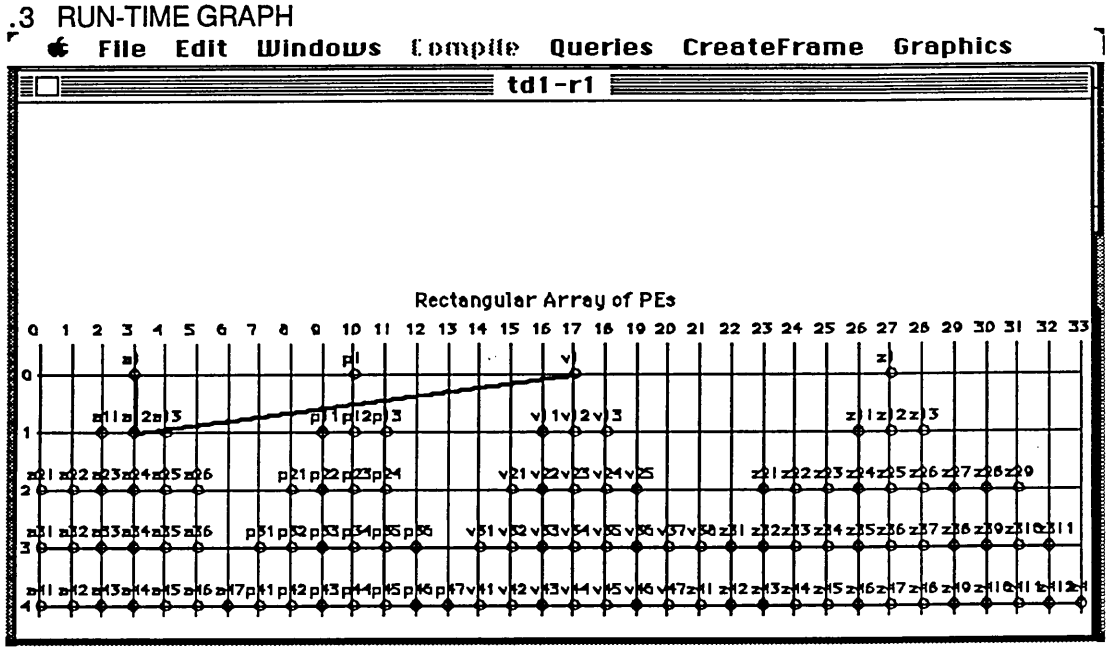
1.2 TIME-LIST :

(v1 (0 9)(0 0)(0 .8))

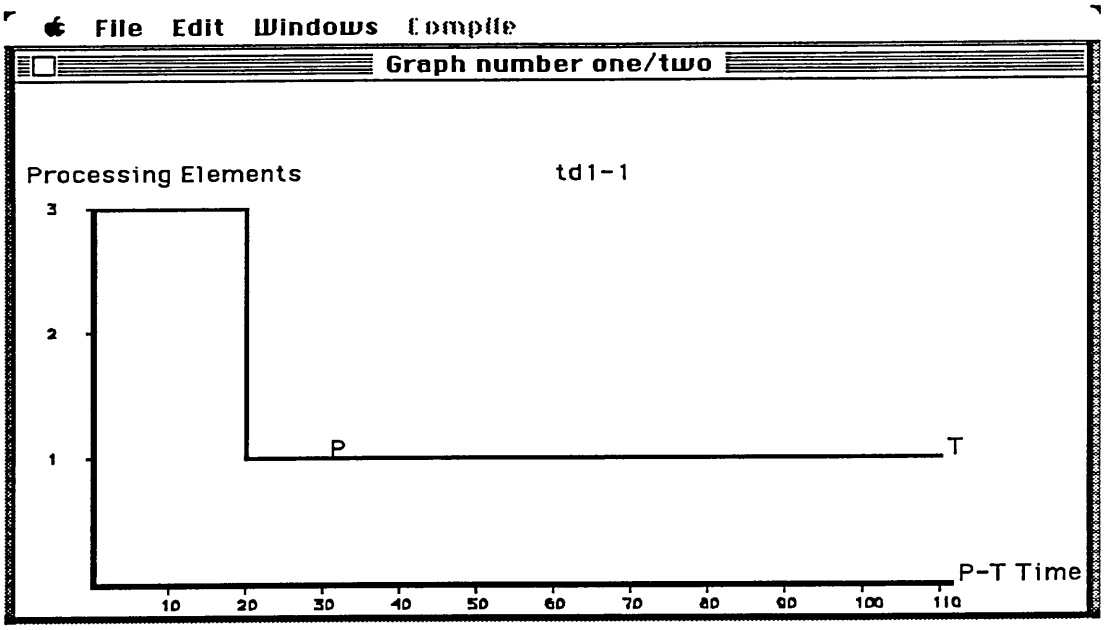
(a1 (0 0)(0 0)(0 2.8))

(a12 (1 1)(0 115.2)(2.8 35.1))

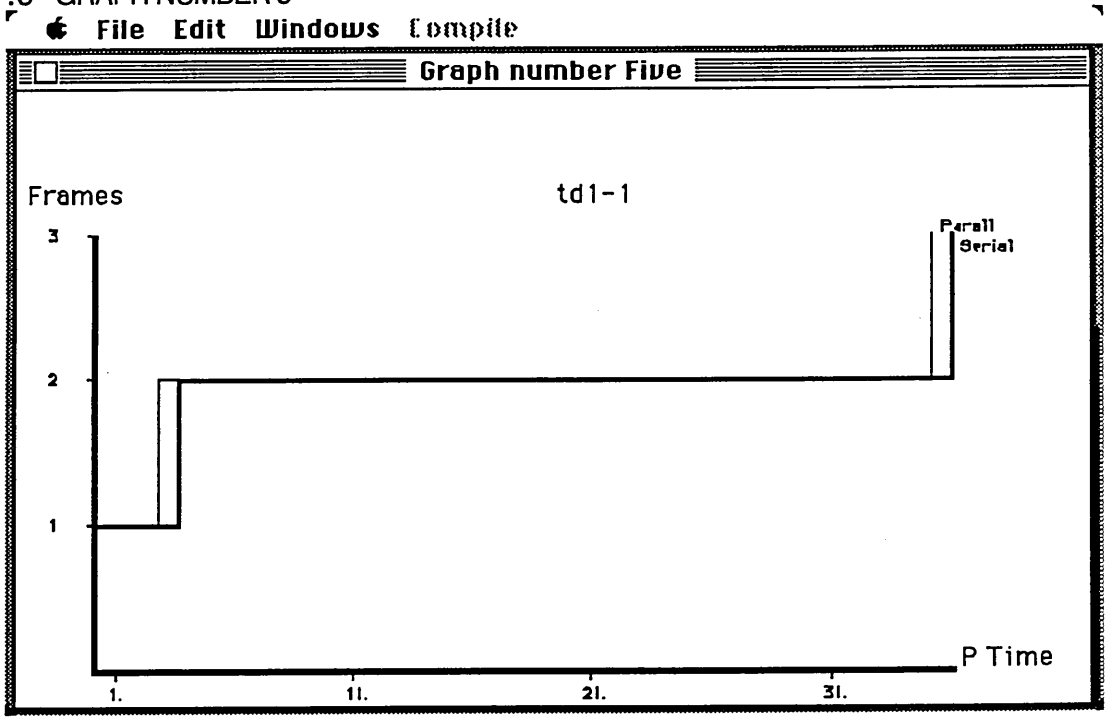
1.3 RUN-TIME GRAPH



1.4 GRAPH NUMBER 1/2



1.5 GRAPH NUMBER 5



2.1 QUERY OBJECT : P22

2.2 TIME-LIST :

(v1 (0 9)(0 0)(0 .8))

(a1 (0 0)(0 0)(0 2.8))

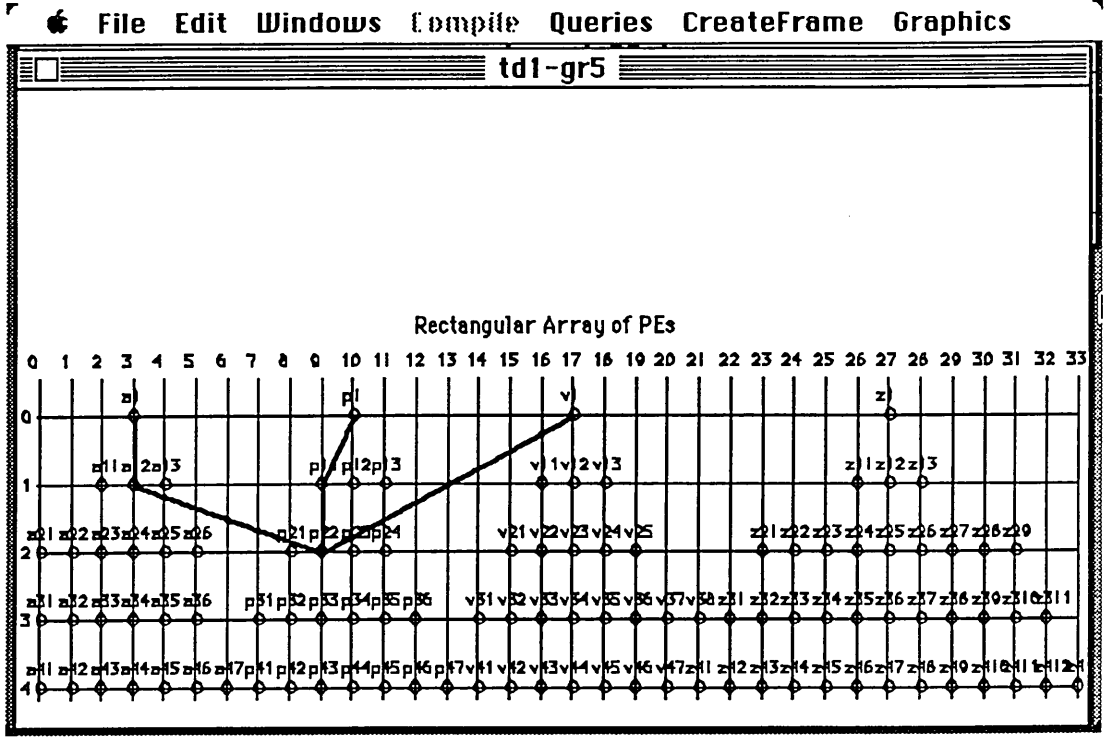
(a12 (1 1)(0 176.4)(2.8 35.1))

(p1 (4 8)(0 0)(0 1.6))

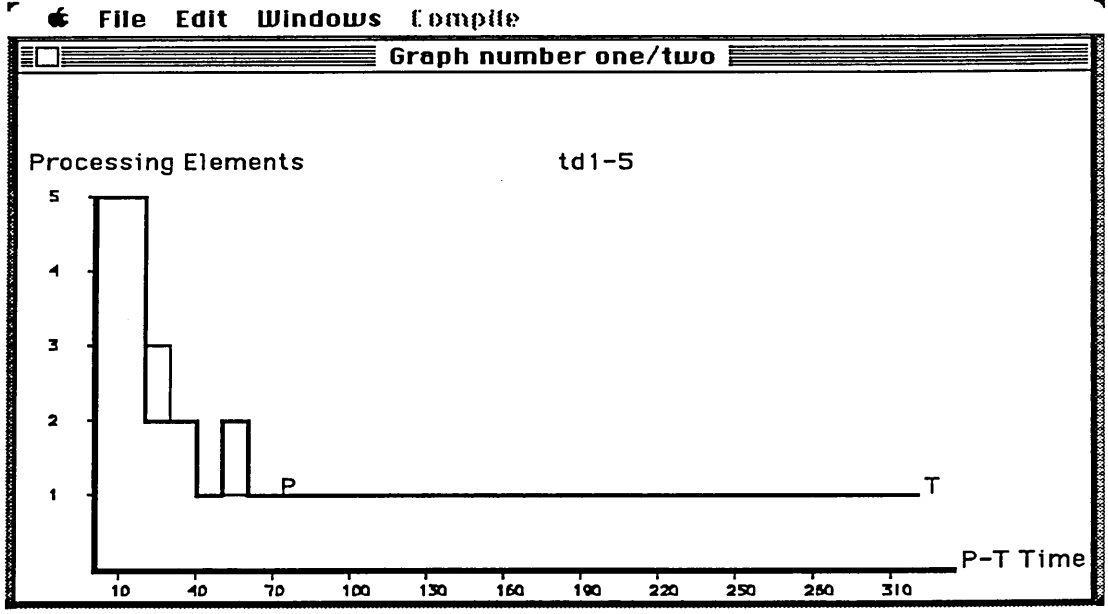
(p11 (3 7)(0 44.4)(1.6 10.2))

(p22 (2 8)(44.4 322.)(10.2 77.6))

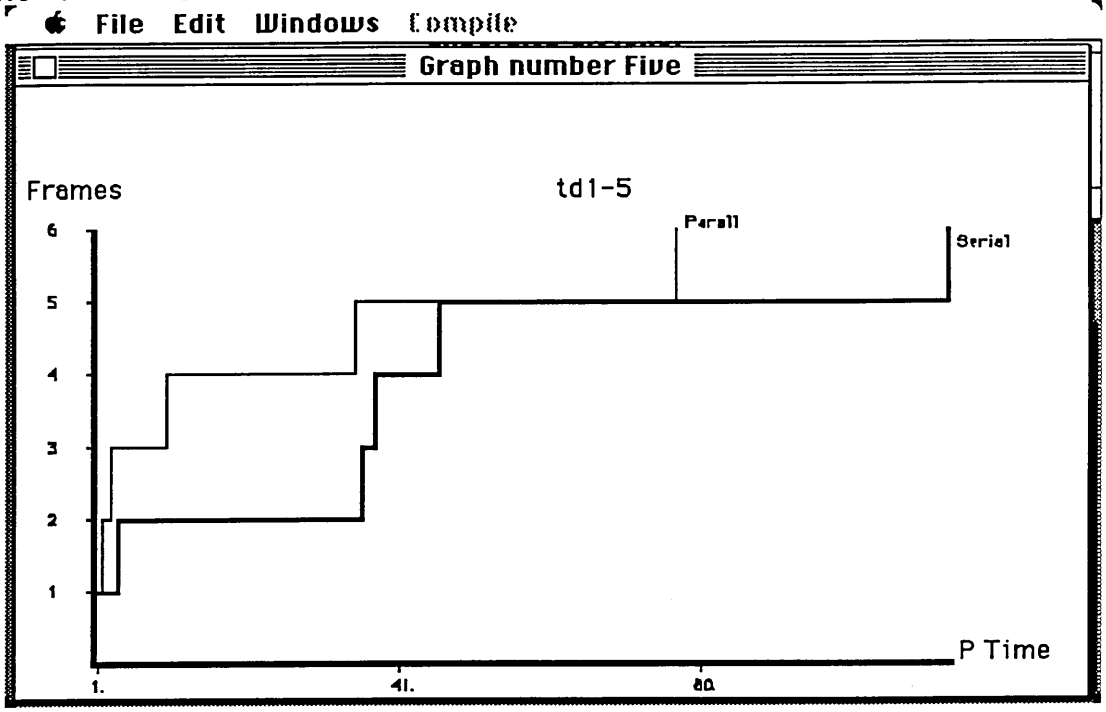
2.3 RUN-TIME GRAPH



2.4 GRAPH NUMBER 1/2



2.5 GRAPH NUMBER 5

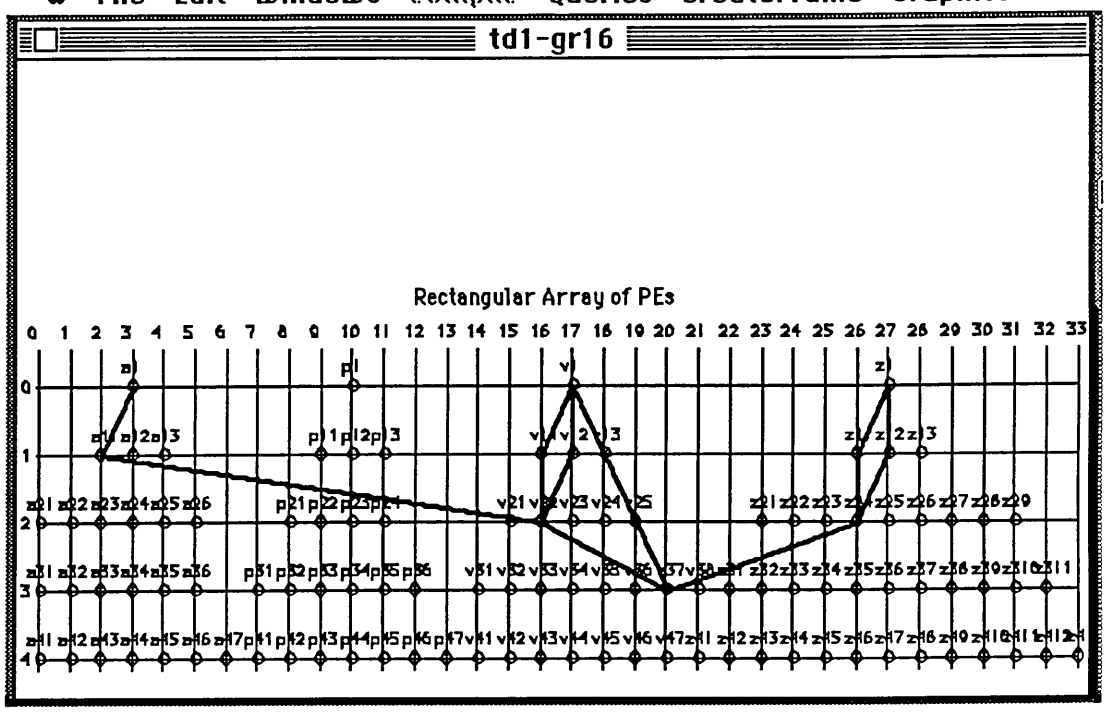


3.1 QUERY OBJECT : v37

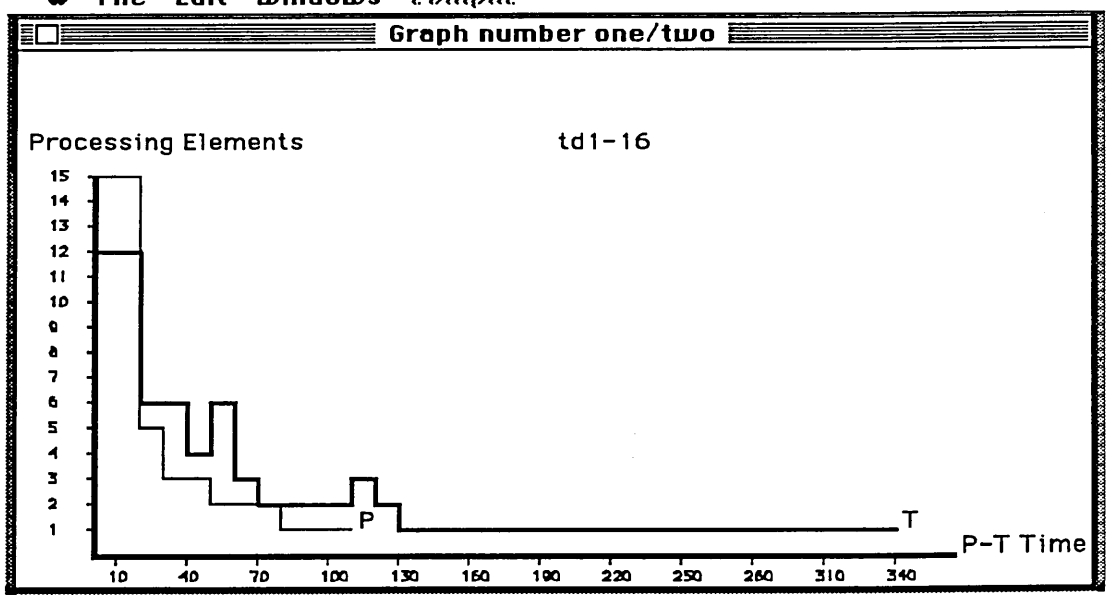
3.2 TIME-LIST :

(z1 (0 27)(0 0)(0 .8))
(z11 (1 26)(0 47.2)(.8 3.7))
(z1 (0 27)(0 0)(0 .8))
(z12 (1 27)(0 23.6)(.8 3.7))
(z24 (2 26)(23.6 76.4)(3.7 17.9))
(a1 (0 3)(0 0)(0 2.8))
(a11 (1 2)(0 59.6)(2.8 30.8))
(v1 (0 17)(0 0)(0 .8))
(v12 (1 17)(0 23.6)(.8 3.7))
(v1 (0 17)(0 0)(0 .8))
(v11 (1 16)(0 47.2)(.8 3.7))
(v22 (2 16)(47.2 551.2)(3.7 63.1))
(v1 (0 17)(0 0)(0 .8))
(v13 (1 18)(0 47.2)(.8 3.7))
(v25 (2 19)(47.2 100.2)(3.7 11.9))
(v37 (3 20)(100.2 333.2)(11.9 116.7))

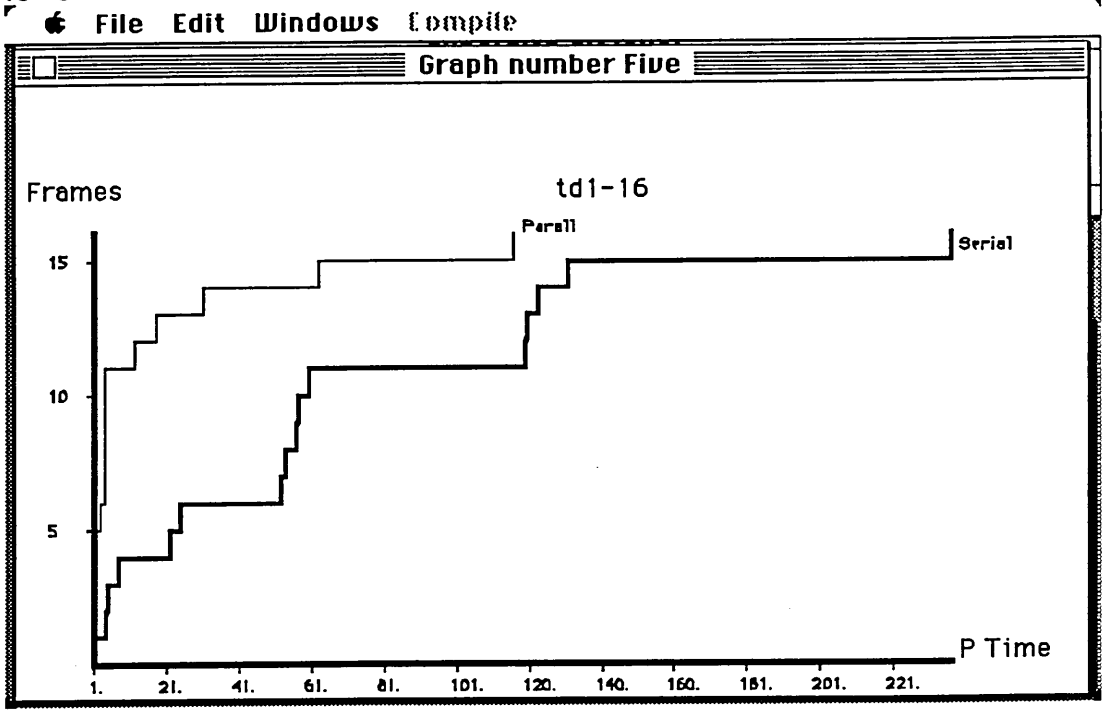
3.3 RUN-TIME GRAPH



3.4 GRAPH NUMBER 1/2



3.5 GRAPH NUMBER 5



4.1 QUERY OBJECT : v44

4.2 TIME-LIST :

(v1 (0 17)(0 0)(0 .8))

(v13 (1 18)(0 56.8)(.8 3.7))

(v24 (2 18)(56.8 88.1)(3.7 11.9))

(v1 (0 17)(0 0)(0 .8))

(a1 (0 3)(0 0)(0 2.8))

(a12 (1 3)(0 426.)(2.8 35.1))

(p1 (0 10)(0 0)(0 1.6))

(p11 (1 9)(0 62.)(1.6 10.2))

(p22 (2 9)(62. 366.5)(10.2 77.6))

(p1 (0 10)(0 0)(0 1.6))

(p13 (1 11)(0 62.)(1.6 10.2))

(p24 (2 11)(62. 97.8)(10.2 30.))

(p35 (3 11)(97.8 378.6)(30. 123.))

(a1 (0 3)(0 0)(0 2.8))

(a11 (1 2)(0 69.2)(2.8 30.8))

(v1 (0 17)(0 0)(0 .8))

(v12 (1 17)(0 28.4)(.8 3.7))

(v1 (0 17)(0 0)(0 .8))

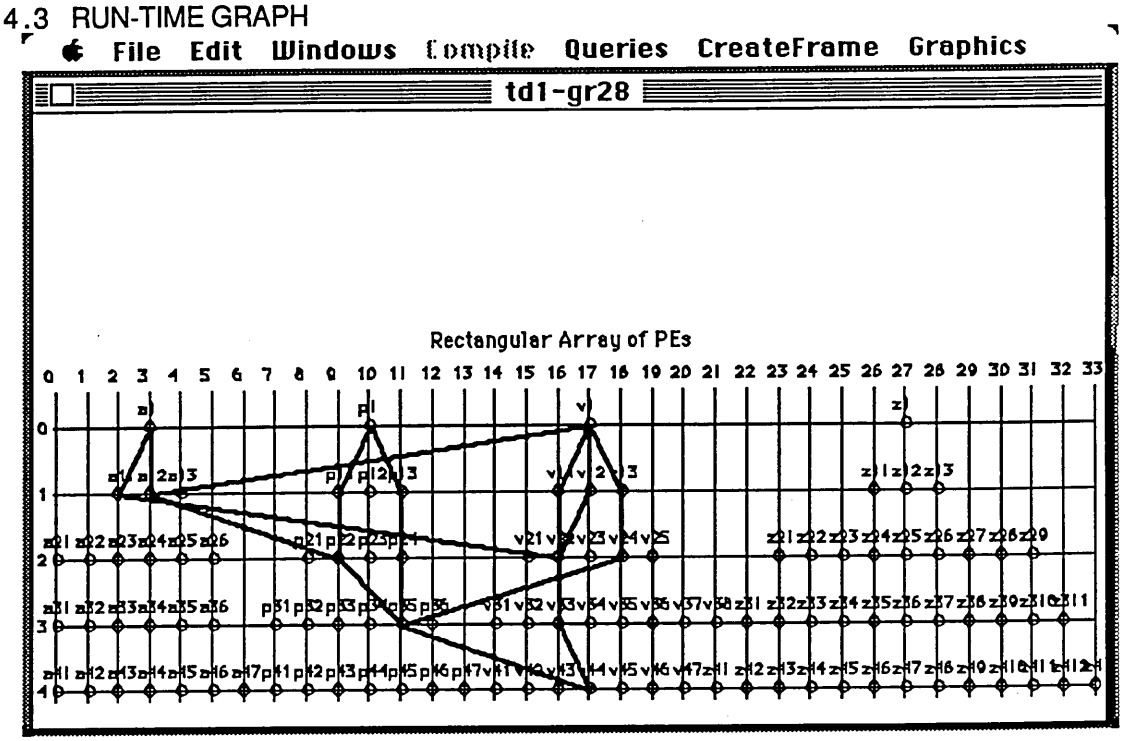
(v11 (1 16)(0 56.8)(.8 3.7))

(v22 (2 16)(56.8 632.8)(3.7 63.1))

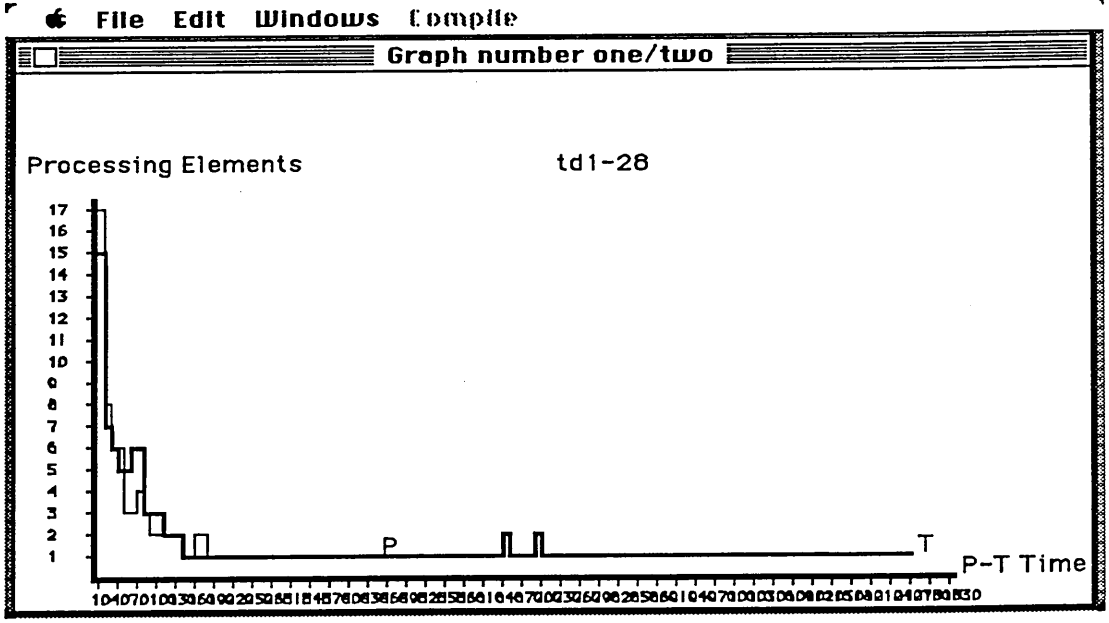
(v33 (3 16)(632.8 684.2)(63.1 160.))

(v44 (4 17)(684.2 1272.9)(160. 440.1))

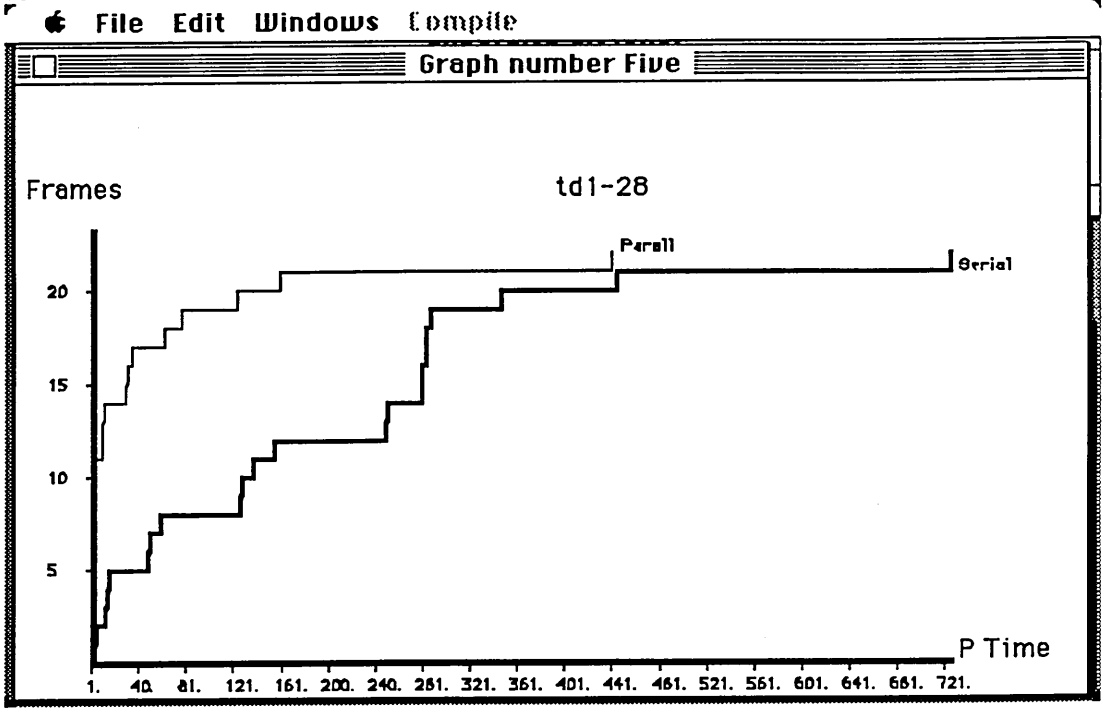
4.3 RUN-TIME GRAPH



4.4 GRAPH NUMBER 1/2



4.5 GRAPH NUMBER 5



5.1 QUERY OBJECT : v47

5.2 TIME-LIST :

(p1 (0 10)(0 0)(0 1.6))

(p12 (1 10)(0 29.5)(1.6 10.2))

(z1 (0 27)(0 0)(0 .8))

(z11 (1 26)(0 53.8)(.8 3.7))

(z21 (2 23)(53.8 520.)(3.7 26.9))

(z31 (3 22)(520. 604.8)(26.9 72.4))

(a1 (0 3)(0 0)(0 2.8))

(a13 (1 4)(0 66.2)(2.8 30.8))

(a1 (0 3)(0 0)(0 2.8))

(a11 (1 2)(0 66.2)(2.8 30.8))

(a21 (2 0)(66.2 250.7)(30.8 112.4))

(v1 (0 17)(0 0)(0 .8))

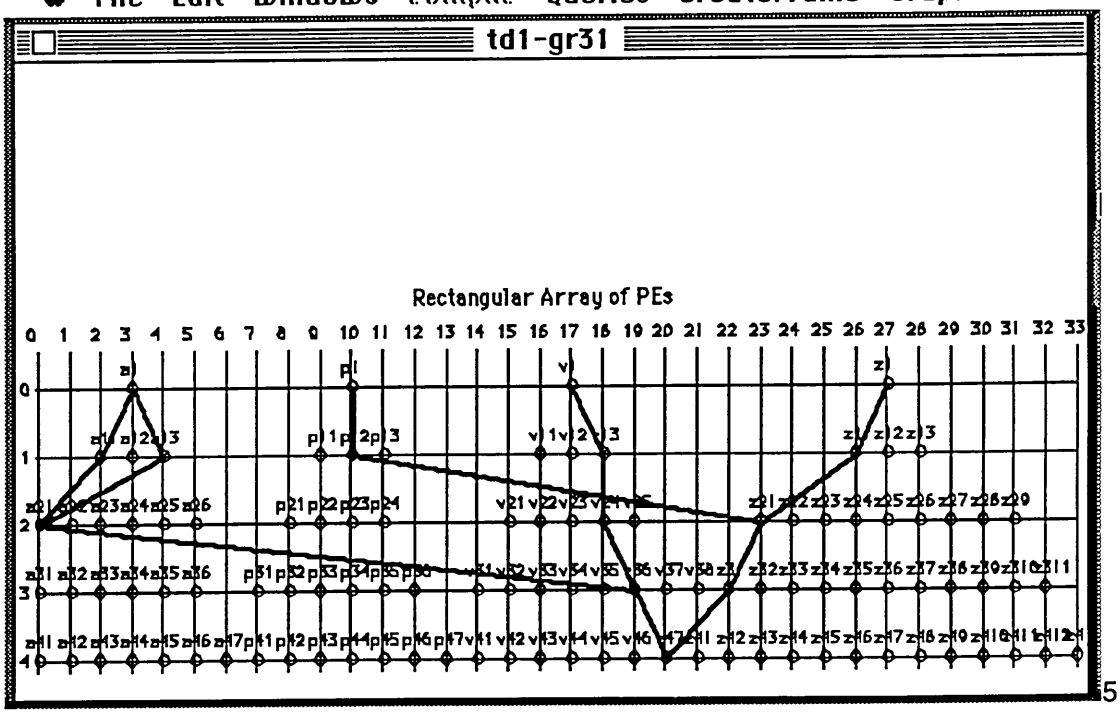
(v13 (1 18)(0 53.8)(.8 3.7))

(v24 (2 18)(53.8 83.6)(3.7 11.9))

(v36 (3 19)(83.6 1083.6)(11.9 86.2))

(v47 (4 20)(1083.6 1225.2)(86.2 284.9))

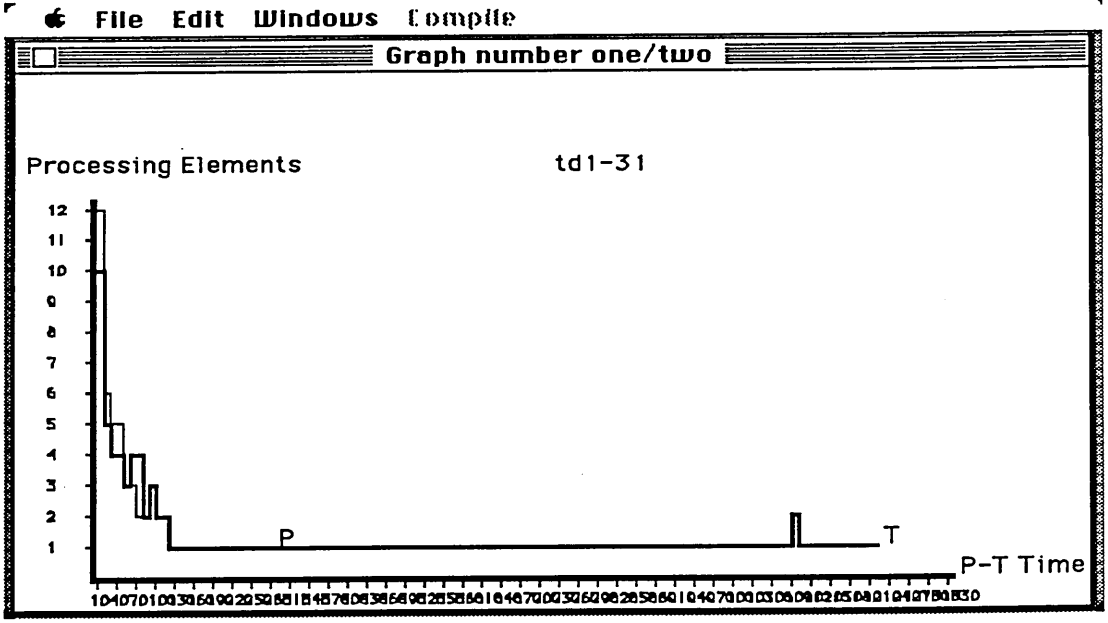
5.3 RUN-TIME GRAPH



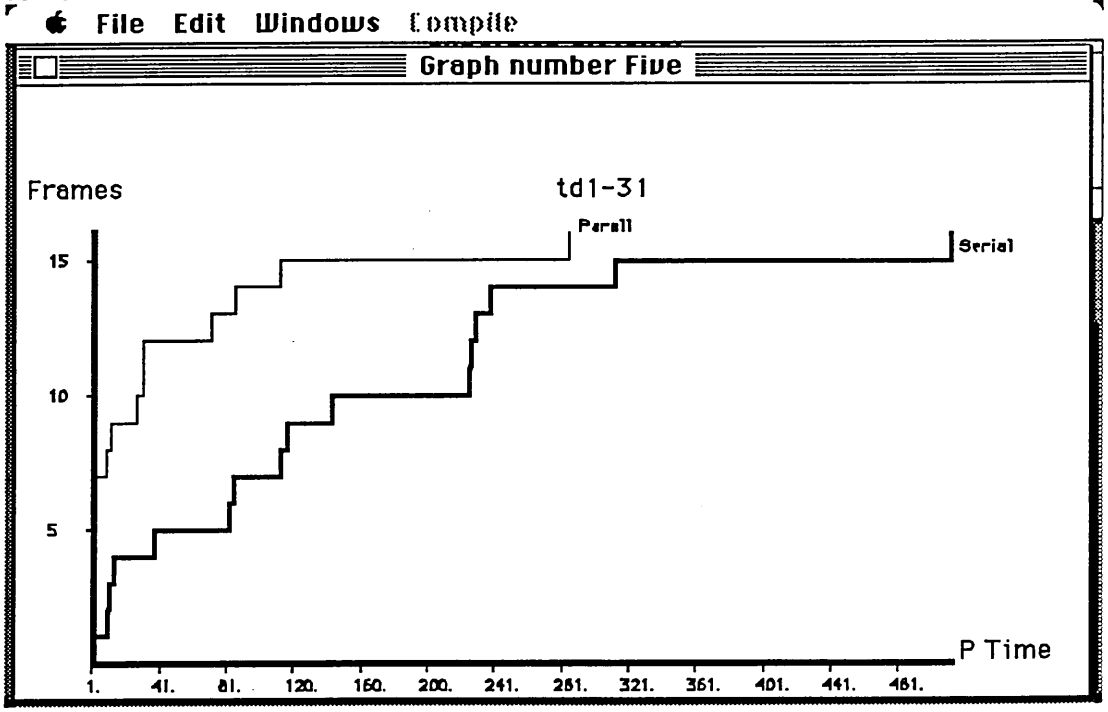
5.

4

GRAPH NUMBER 1/2



5.5 GRAPH NUMBER 5



D2.0 QUERIES OF TYPE QUERY-OBJECT NUMBER TWO

In this type of query a query-frame is created at each interrogation. Each query is a conjunction of two or more slot-names with/without values. The missing slots/values are represented by variables which in the course of propagation will be bound to appropriate values.

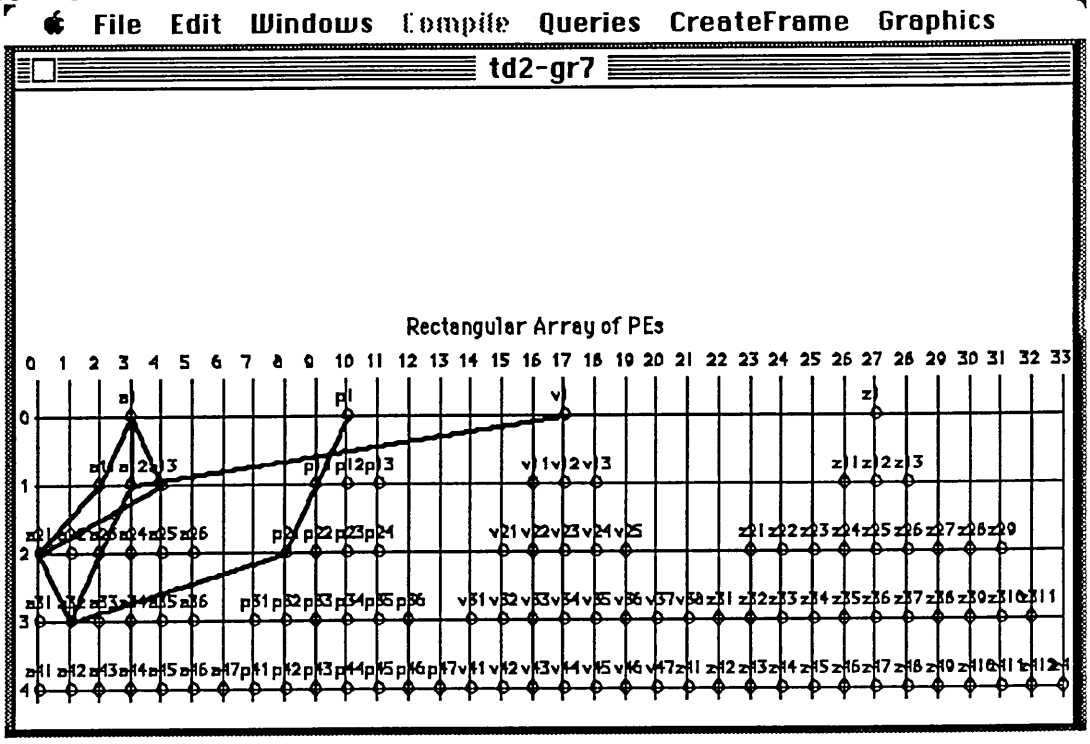
1.1 QUERY FRAME:

```
((783   a32)  ((a1_inh_slot1  (value  ?V1))
              (a1_inh_slot2  (value  ?V2))
              (a11_inh_slot1  (value  ?V3))
              (v1_inh_slot1  (value  ?V4))
              (p1_inh_slot1  (value  ?V5))  ))
```

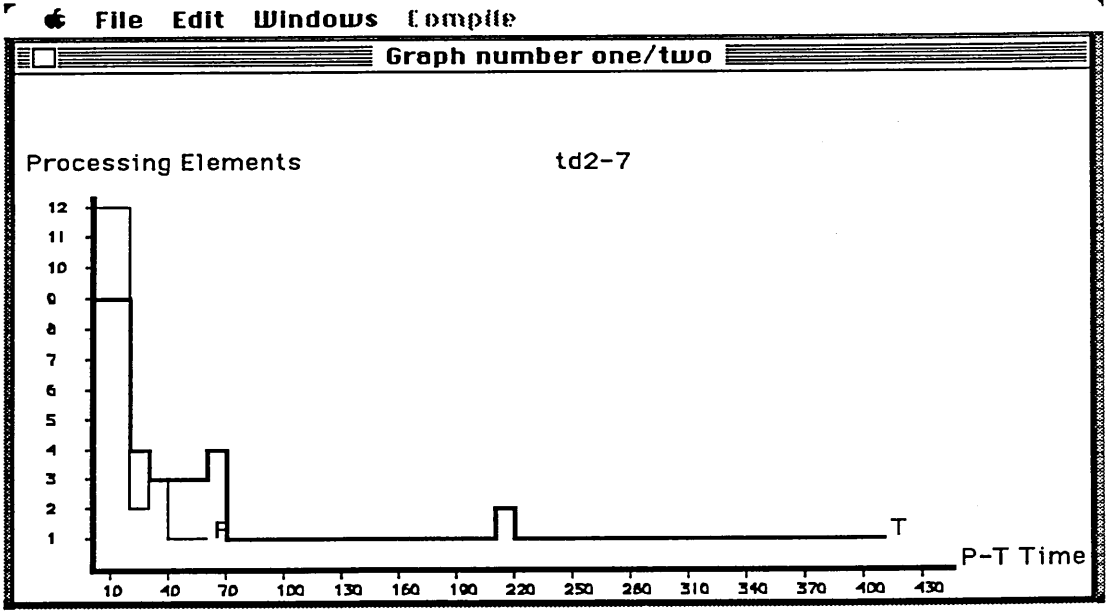
1.2 TIME-LIST :

```
(p1  (0   10 )(0   0 )(0   .8 ))
(p11  (1   9 )(0   47.6 )(.8   3.1 ))
(p21  (2   8 )(47.6   97.2 )(3.1   6.9 ))
(v1  (0   17 )(0   0 )(0   .8 ))
(a1  (0   3 )(0   0 )(0   1.4 ))
(a12  (1   3 )(0   357. )(1.4   9.3 ))
(a23  (2   2 )(357.   417.4 )(9.3   23.7 ))
(a1  (0   3 )(0   0 )(0   1.4 ))
(a13  (1   4 )(0   51.2 )(1.4   8.4 ))
(a1  (0   3 )(0   0 )(0   1.4 ))
(a11  (1   2 )(0   51.2 )(1.4   8.4 ))
(a21  (2   0 )(51.2   200.7 )(8.4   27.4 ))
(a32  (3   1 )(200.7   407.1 )(27.4   66.5 ))
```

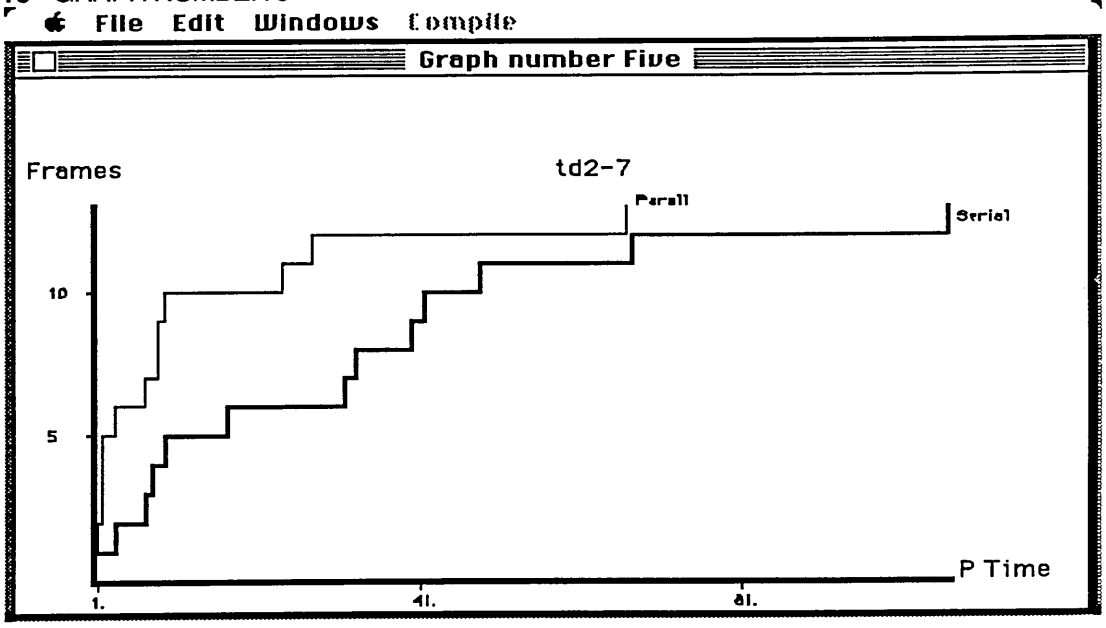
1.3 RUN-TIME GRAPH



1.4 GRAPH NUMBER 1/2



1.5 GRAPH NUMBER 5



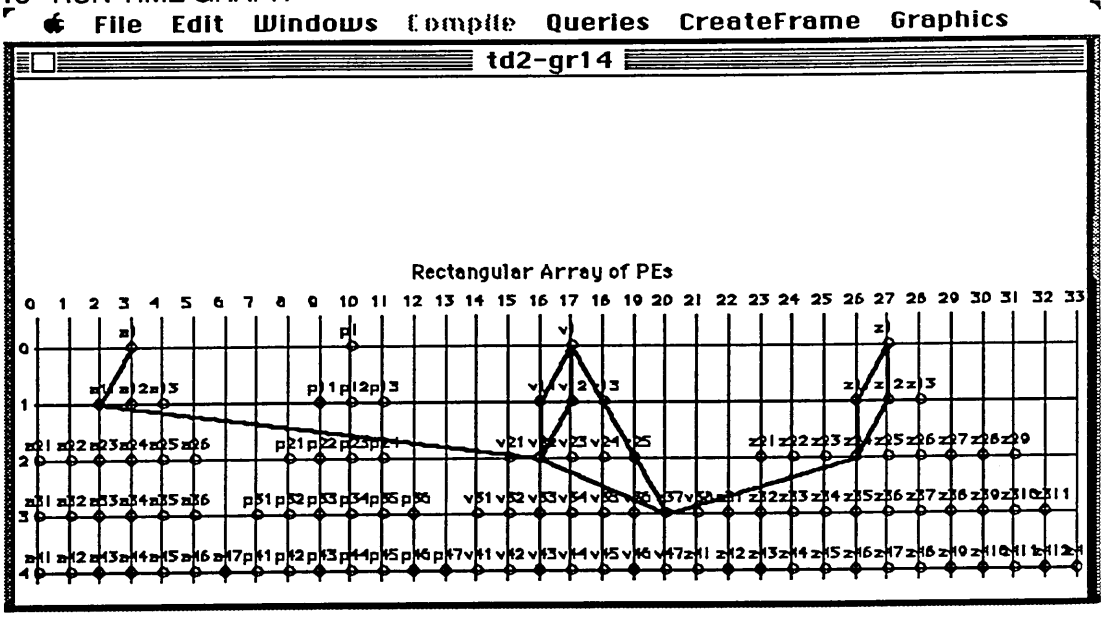
2.1 QUERY FRAME :

```
((1383    v37)      ((z12_inh_slot1  (value  ?V1))
                    (z11_inh_slot1  (value  ?V2))
                    (a1_inh_slot4  (value  ?V3))
                    (v1_inh_slot1  (value  ?V4))  ))
```

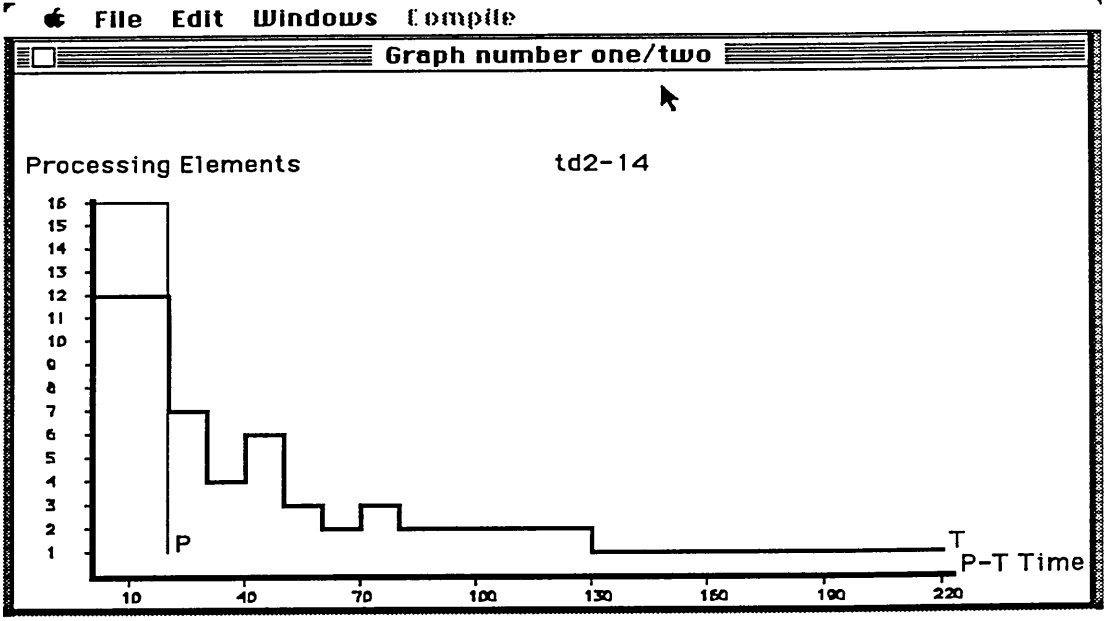
2.2 TIME-LIST :

```
(z1  (0   27  )(0   0  )(0   0  ))
(z11  (1   26  )(0   29.  )(0   .8  ))
(z1  (0   27  )(0   0  )(0   0  ))
(z12  (1   27  )(0   14.5  )(0   .9  ))
(z24  (2   26  )(14.5   49.1  )(.9   5.8  ))
(a1  (0   3  )(0   0  )(0   .7  ))
(a11  (1   2  )(0   33.4  )(.7   2.9  ))
(v1  (0   17  )(0   0  )(0   .8  ))
(v12  (1   17  )(0   17.1  )(.8   3.1  ))
(v1  (0   17  )(0   0  )(0   .8  ))
(v11  (1   16  )(0   34.2  )(.8   3.1  ))
(v22  (2   16  )(34.2   277.2  )(3.1   9.  ))
(v1  (0   17  )(0   0  )(0   .8  ))
(v13  (1   18  )(0   34.2  )(.8   3.1  ))
(v25  (2   19  )(34.2   68.6  )(3.1   6.9  ))
(v37  (3   20  )(68.6   214.9  )(6.9   21.8  ))
```

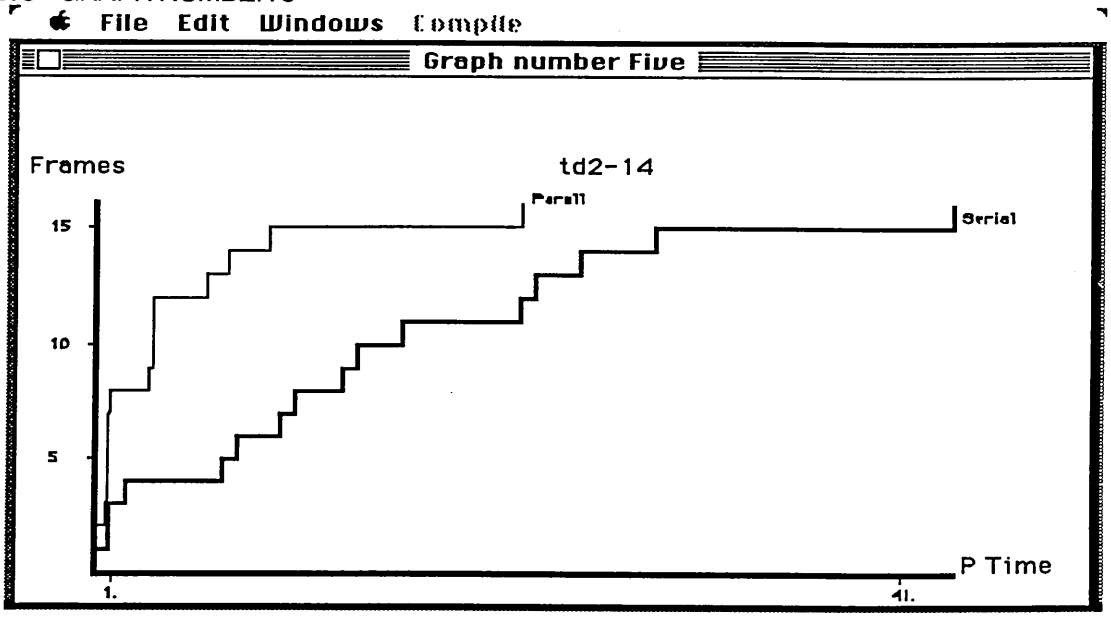
2.3 RUN-TIME GRAPH



2.4 GRAPH NUMBER 1/2



2.5 GRAPH NUMBER 5



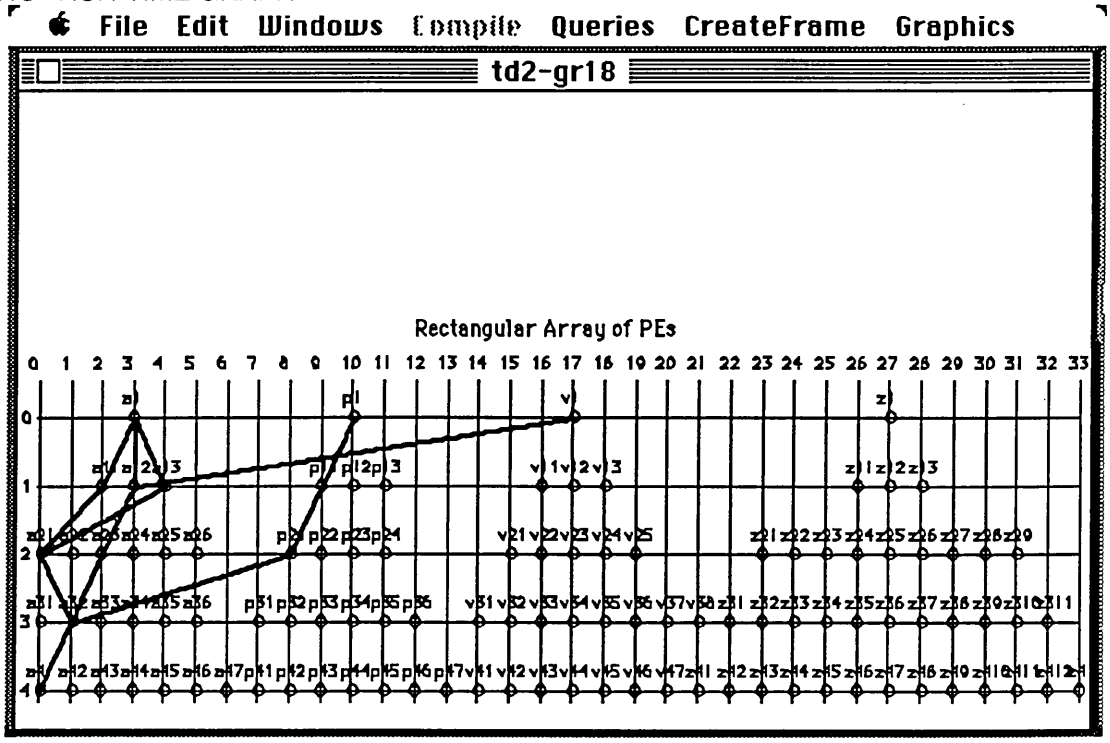
3.1 QUERY FRAME :

```
((684   a41)  ((a1_inh_slot1  (value  ?V1))
                (a1_inh_slot2  (value  ?V2))
                (a1_inh_slot3  (value  ?V3))
                (v1_inh_slot1  (value  ?V4))
                (p1_inh_slot1  (value  ?V5))
                (p21_inh_slot1  (value  ?V6))
                (a32_inh_slot1  (value  ?V7))))
```

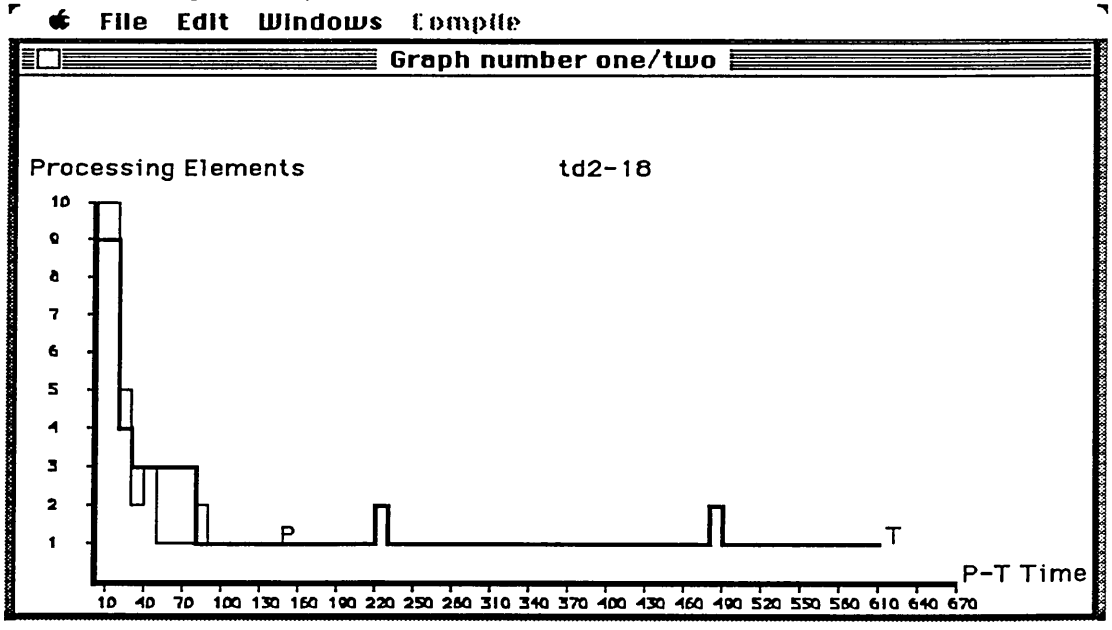
3.2 TIME-LIST :

```
(p1  (0   10 )(0   0 )(0   .8 ))        (p11  (1   9 )(0   53.2 )(.8   3.1 ))
(p21  (2   8 )(53.2   108.4 )(3.1   7.4 ))   (v1  (0   17 )(0   0 )(0   .8 ))
(a1  (0   3 )(0   0 )(0   2.1 ))        (a12  (1   3 )(0   399. )(2.1   15.3 ))
(a23  (2   2 )(399.   471.4 )(15.3   39.6 ))        (a1  (0   3 )(0   0 )(0   2.1 ))
(a13  (1   4 )(0   61.2 )(2.1   12.9 ))              (a1  (0   3 )(0   0 )(0   2.1 ))
(a11  (1   2 )(0   61.2 )(2.1   12.9 ))              (a21  (2   0 )(61.2   219.2 )(12.9
32.4 ))
(a32  (3   1 )(219.2   470.4 )(32.4   78.2 ))       (a41  (4   0 )(470.4   615.2
)(78.2   144.1 ))
```

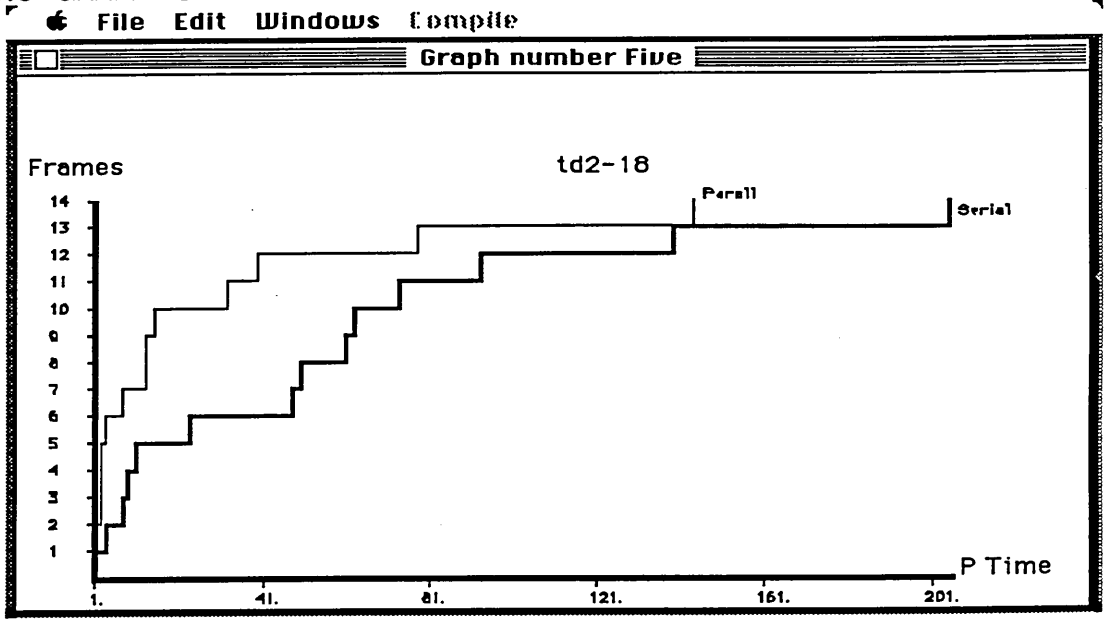
3.3 RUN-TIME GRAPH



3.4 GRAPH NUMBER 1/2



3.5 GRAPH NUMBER 5



4.1 QUERY FRAME :

```
((784   a42) ((a1_inh_slot1 (value ?V1))      (a1_inh_slot2 (value ?V5))
              (a1_inh_slot4 (value ?V2))   (a21_inh_slot1 (value ?V6))
              (a12_inh_slot1 (value ?V3)) (p1_inh_slot1 (value ?V7))
              (z1_inh_slot1 (value ?V4))  (z31_inh_slot1 (value ?V8))
                                          (a42_inh_slot1 (value ?V9)) ))
```

4.2 TIME-LIST :

```
(p1 (0   10 )(0   0 )(0   .8 ))
(p12 (1   10 )(0   34.1 )(.8   3.1 ))
(z1 (0   27 )(0   0 )(0   .8 ))
(z11 (1   26 )(0   68.2 )(.8   3.1 ))
(z21 (2   23 )(68.2   559.6 )(3.1   8.7 ))
(z31 (3   22 )(559.6   633.2 )(8.7   16.7 ))
(p1 (0   10 )(0   0 )(0   .8 ))
(p11 (1   9 )(0   68.2 )(.8   3.1 ))
(p21 (2   8 )(68.2   138.4 )(3.1   6.9 ))
(a1 (0   3 )(0   0 )(0   2.1 ))
(a12 (1   3 )(0   38.1 )(2.1   14.7 ))
(a23 (2   2 )(38.1   122.9 )(14.7   39.5 ))
(a1 (0   3 )(0   0 )(0   2.1 ))
```

(a13 (1 4)(0 76.2)(2.1 12.9))

(a1 (0 3)(0 0)(0 2.1))

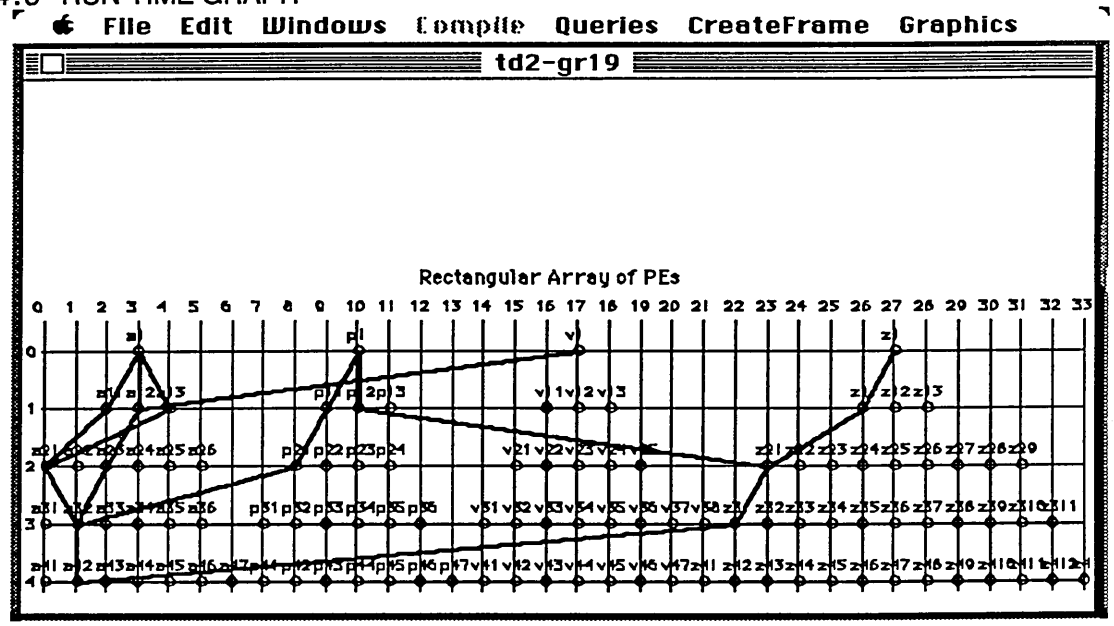
(a11 (1 2)(0 76.2)(2.1 10.))

(a21 (2 0)(76.2 274.2)(10. 31.))

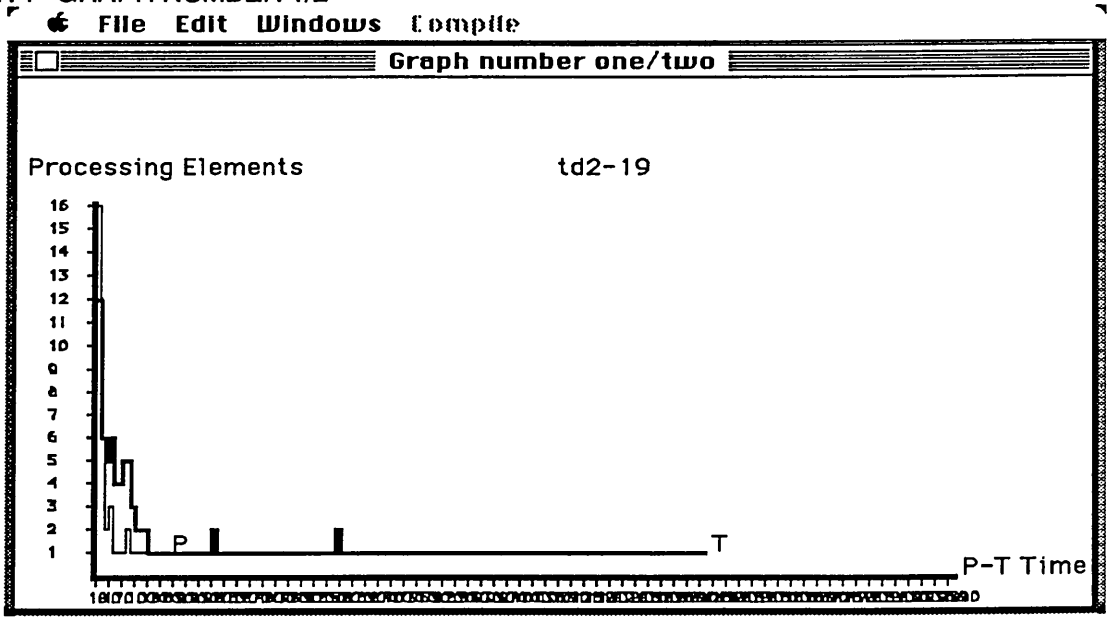
(a32 (3 1)(274.2 563.)(31. 79.2))

(a42 (4 1)(563. 1434.2)(79.2 174.2))

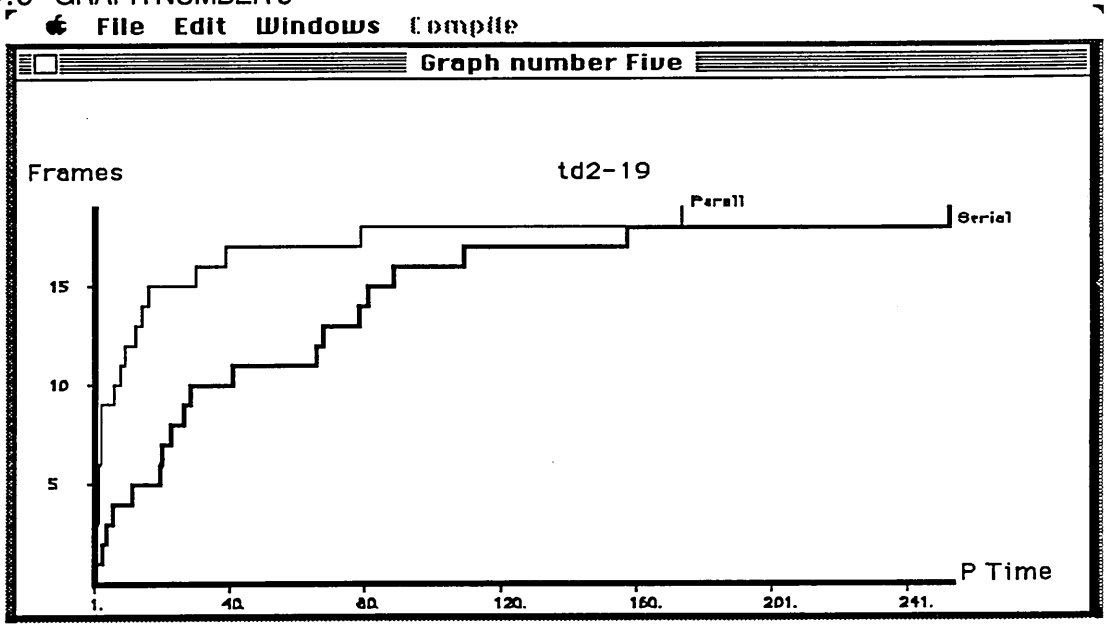
4.3 RUN-TIME GRAPH



4.4 GRAPH NUMBER 1/2



4.5 GRAPH NUMBER 5



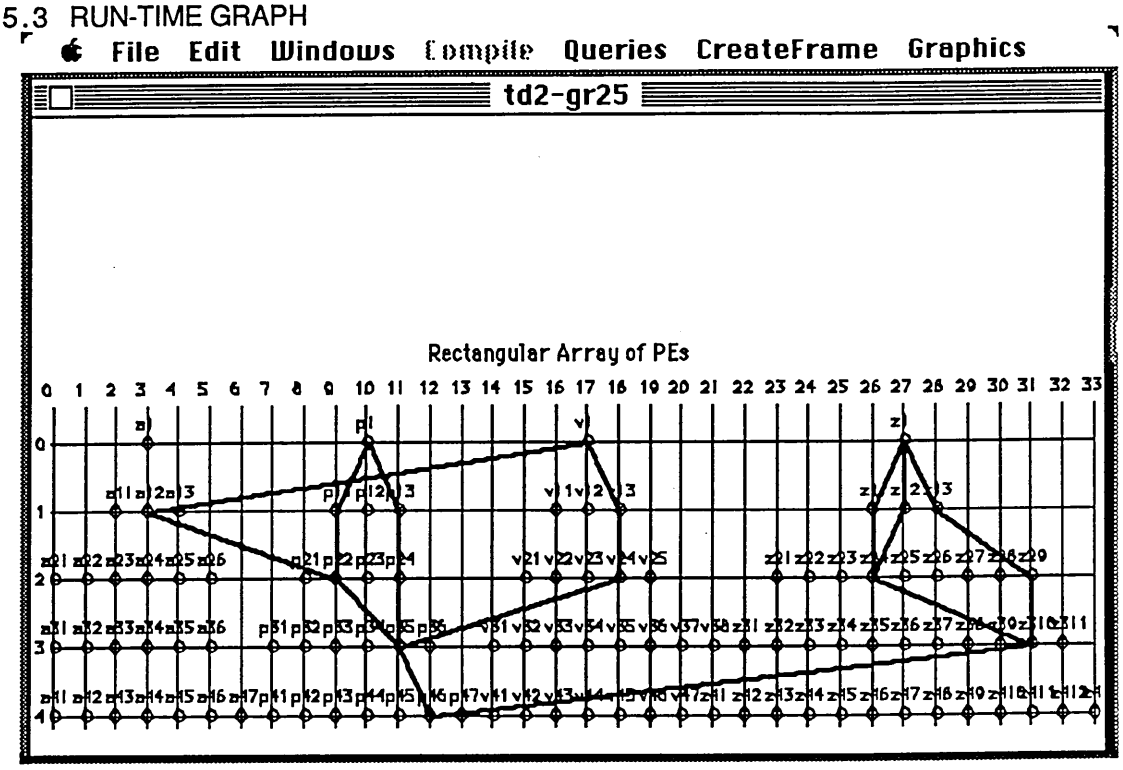
5.1 QUERY FRAME:

```
((1364    p46)      ((p1_inh_slot1  (value  ?V1))
                    (v1_inh_slot1  (value  ?V2))
                    (z1_inh_slot1  (value  ?V3))
                    (z13_inh_slot1  (value  ?V4))
                    (z29_inh_slot1  (value  ?V5))  ))
```

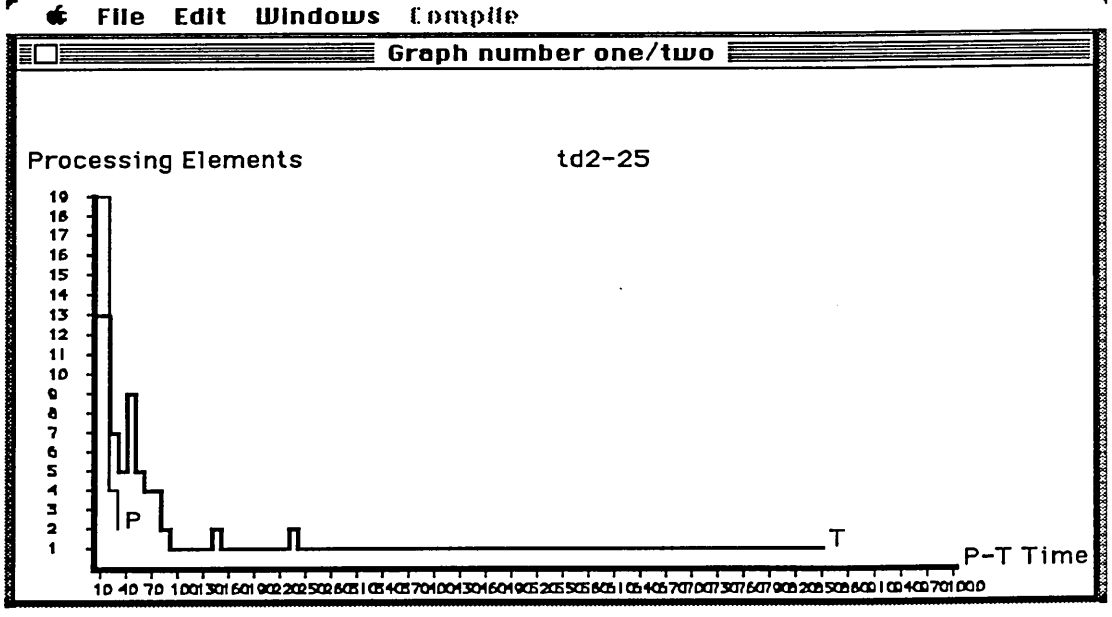
5.2 TIME-LIST :

```
(z1 (0   27 )(0   0 )(0   .8 ))                (z11 (1   26 )(0   39.8 )(.8   3.1 ))
(z1 (0   27 )(0   0 )(0   .8 ))                (z12 (1   27 )(0   19.9 )(.8   3.1 ))
(z24 (2   26 )(19.9   59.7 )(3.1   6.9 ))     (z1 (0   27 )(0   0 )(0   .8 ))
(z13 (1   28 )(0   39.8 )(.8   3.7 ))                 (z29 (2   31 )(39.8   130.6 )(3.7
11.4  ))
(z310 (3   31 )(130.6   265.6 )(11.4   25.6 ))(v1 (0   17 )(0   0 )(0   .8 ))
(v13 (1   18 )(0   39.8 )(.8   3.1 ))                 (v24 (2   18 )(39.8   59.8 )(3.1
6.9  ))
(v1 (0   17 )(0   0 )(0   .8 ))                (p1 (0   10 )(0   0 )(0   .8 ))
(p11 (1   9 )(0   39.8 )(.8   3.1 ))          (p22 (2   9 )(39.8   60.7 )(3.1   7.2 ))
(p1 (0   10 )(0   0 )(0   .8 ))                (p13 (1   11 )(0   39.8 )(.8   3.1 ))
(p24 (2   11 )(39.8   60.7 )(3.1   6.9 ))     (p35 (3   11 )(60.7   220.7 )(6.9   12.2
) )
(p46 (4   12 )(220.7   856.7 )(12.2   32. ))
```

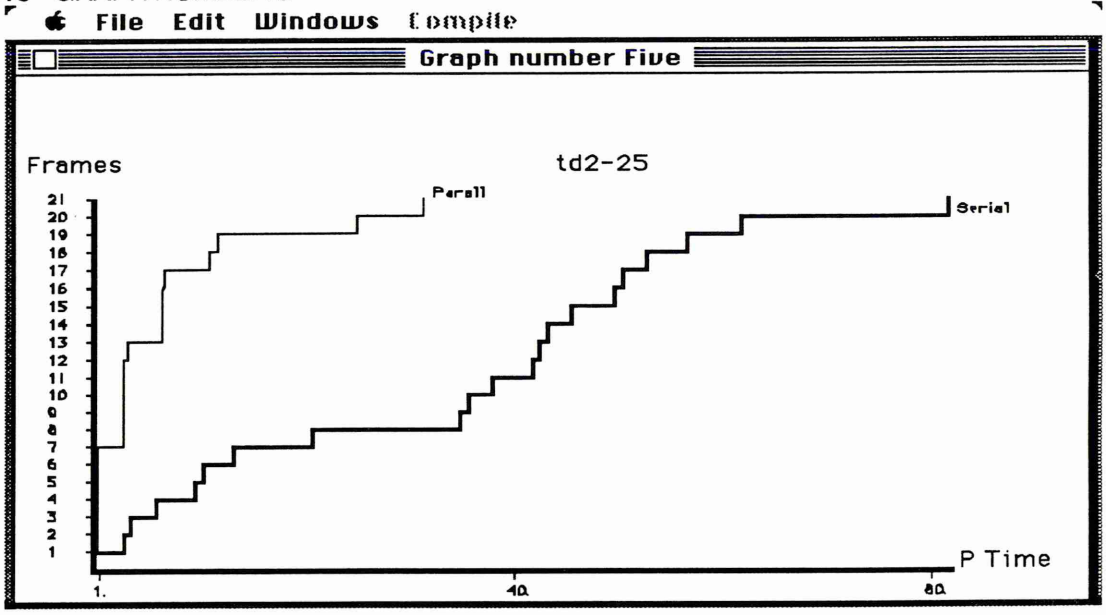
5.3 RUN-TIME GRAPH



5.4 GRAPH NUMBER 1/2



5.5 GRAPH NUMBER 5



D3.0 QUERIES OF TYPE QUERY-OBJECT NUMBER THREE

In this type of query, which is similar to query-object number two, a query-frame is created at each interrogation. Each query is a conjunction of two or more slot-names with/without values. This type of query differs from the previous queries in that there is no frame name given by the user. Instead, for the purpose of the mapping operation, the system asks for the name of hierarchy in which the required frame resides.

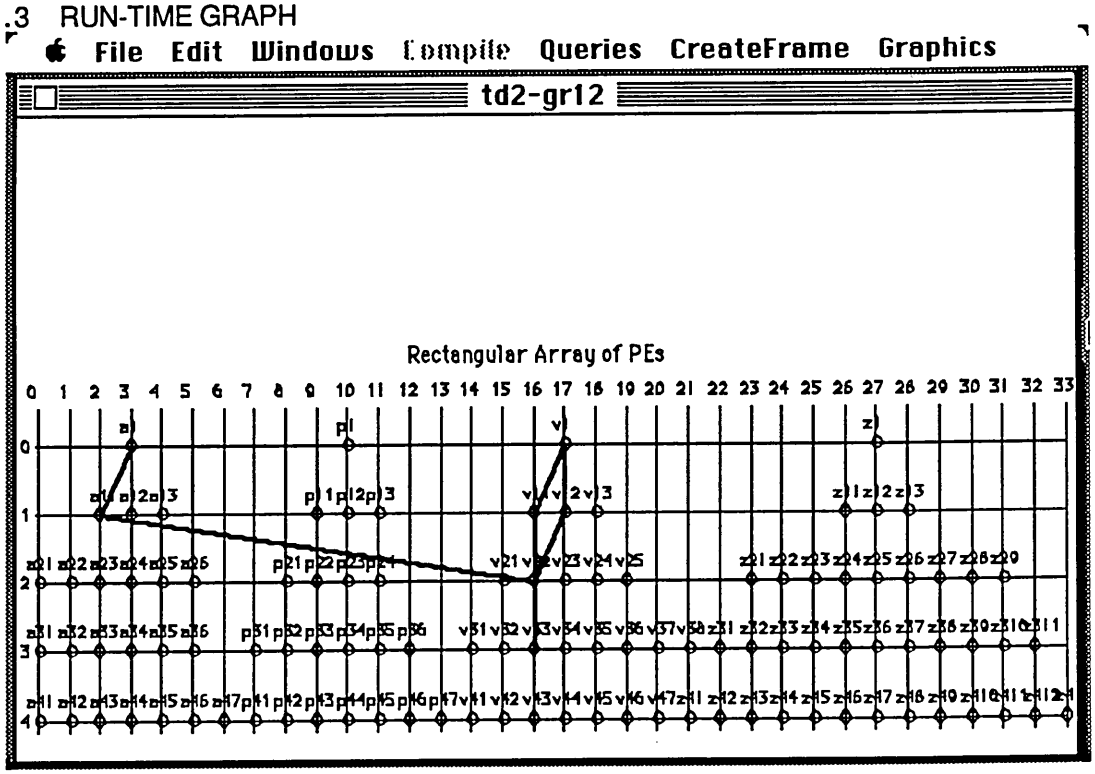
1.1 QUERY FRAME :

```
((V-hir   A-hir)        ;these are the names of hierarchies given by the user
  (?frame-name
          ((a1_inh_slot4  (value  ?V1))
           (a11_inh_slot1  (value  ?V2))
           (v23_inh_slot1  (value  ?V3)) )))
```

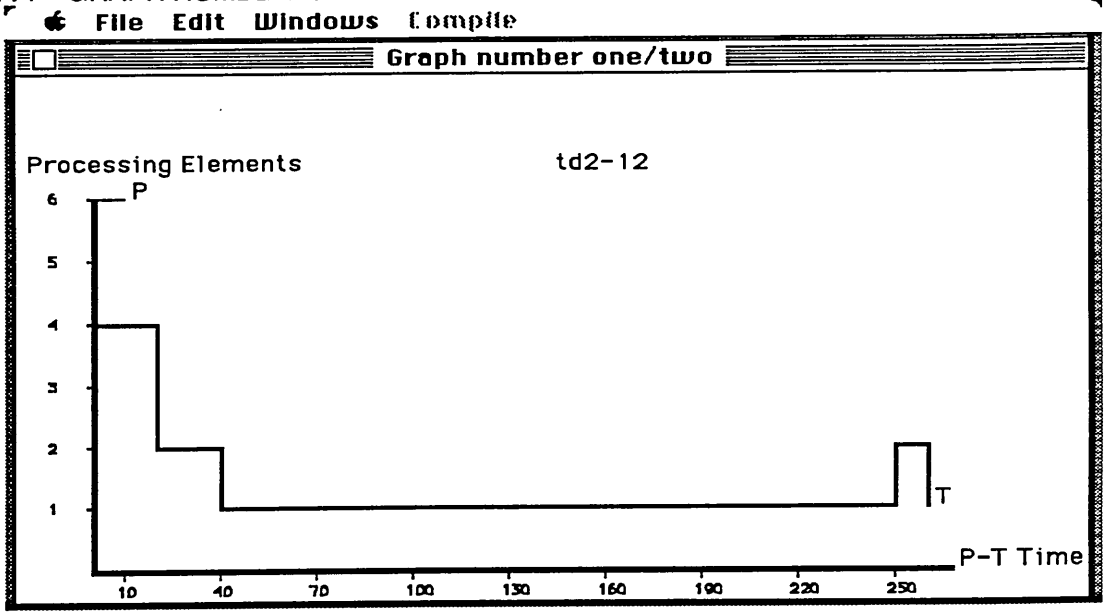
1.2 TIME-LIST :

```
(a1  (0   3 )(0   0 )(0   .7 ))
(a11  (1   2 )(0   25.4 )(.7   3.5 ))
(v1  (0   17 )(0   0 )(0   0 ))
(v11  (1   16 )(0   21. )(0   0 ))
(v22  (2   16 )(21.   246. )(0   2.7 ))
(v33  (3   16 )(246.   266.6 )(2.7   10.5 ))
```

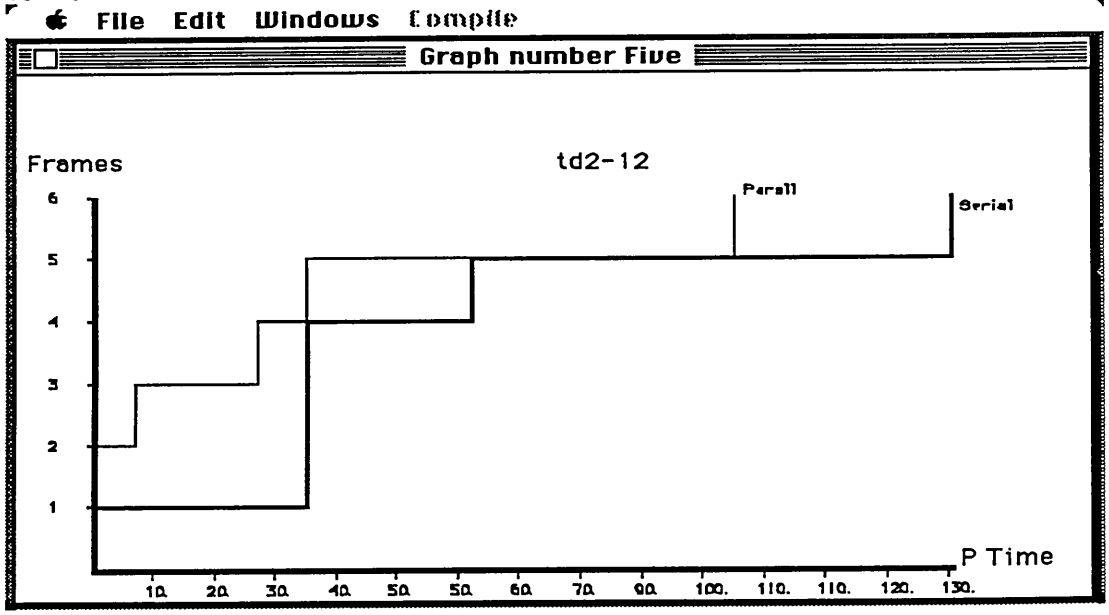
1.3 RUN-TIME GRAPH



1.4 GRAPH NUMBER 1/2



1.5 GRAPH NUMBER 5



2.1 QUERY FRAME :

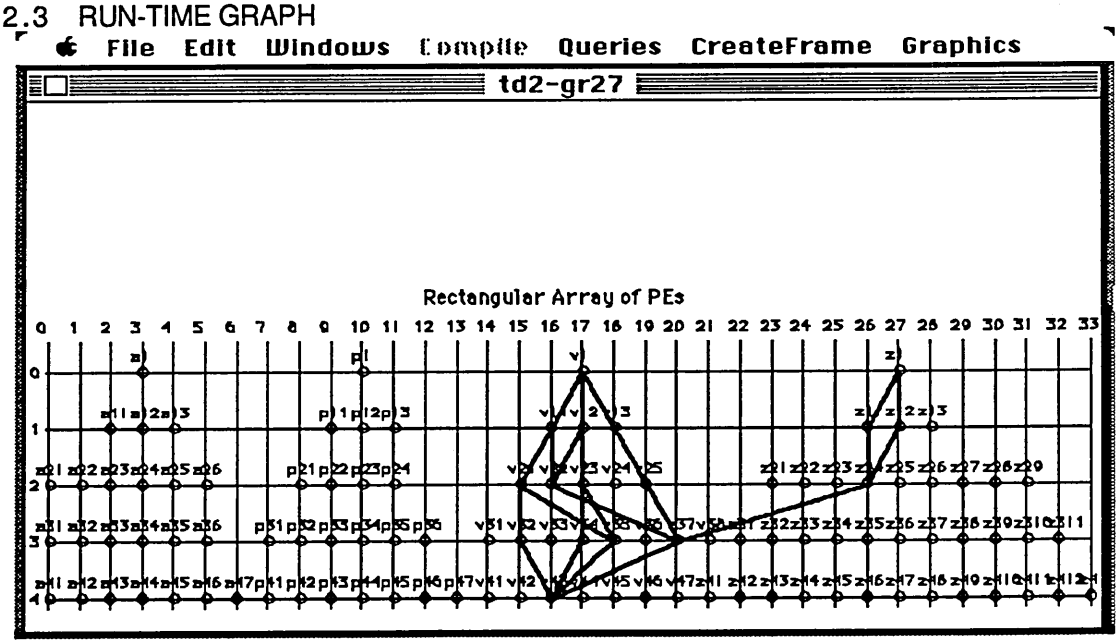
```
((V-hir   Z-hir)        ;these are the names of hierarchies given by the user
  (?frame-name
              ((z12_inh_slot1  (value  ?V1))
              (z1_inh_slot1  (value  ?V2))
              (v12_inh_slot1  (value  ?V3))
              (v1_inh_slot1  (value  ?V4))
              (v11_inh_slot1  (value  ?V5))  )))
```

2.2 TIME-LIST :

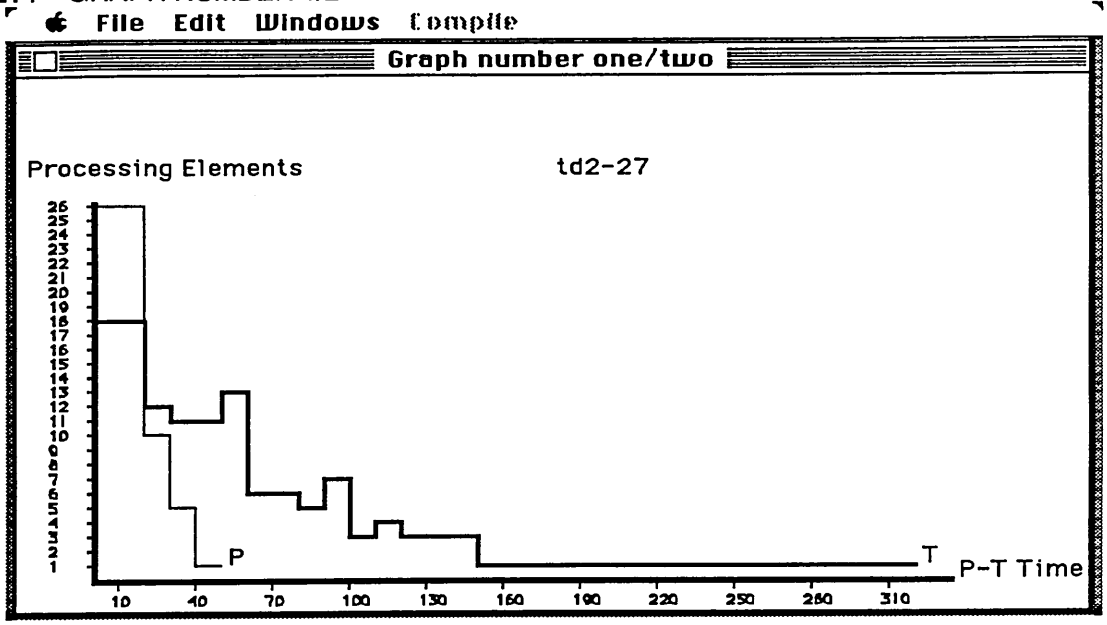
```
(z1  (0   27 )(0   0 )(0   .8 ))                (z11  (1   26 )(0   40. )(.8   3.1 ))
(z1  (0   27 )(0   0 )(0   .8 ))                (z12  (1   27 )(0   20. )(.8   3.7 ))
(z24  (2   26 )(20.   65.6 )(3.7   10.4 ))      (v1  (0   17 )(0   0 )(0   .8 ))
(v12  (1   17 )(0   20. )(.8   3.7 ))           (v1  (0   17 )(0   0 )(0   .8 ))
(v11  (1   16 )(0   40. )(.8   3.7 ))           (v22  (2   16 )(40.   87.6 )(3.7   13.9 ))
(v1  (0   17 )(0   0 )(0   .8 ))                (v13  (1   18 )(0   40. )(.8   3.1 ))
(v25  (2   19 )(40.   80.2 )(3.1   6.9 ))       (v37  (3   20 )(80.2   258. )(6.9   26.5
) )
(v1  (0   17 )(0   0 )(0   .8 ))                (v11  (1   16 )(0   40. )(.8   3.7 ))
(v21  (2   15 )(40.   85.6 )(3.7   10.4 ))      (v1  (0   17 )(0   0 )(0   .8 ))
(v12  (1   17 )(0   20. )(.8   3.7 ))           (v23  (2   17 )(20.   43.8 )(3.7   10.4 ))
(v35  (3   18 )(43.8   135. )(10.4   24.4 ))        (v1  (0   17 )(0   0 )(0   .8 ))
(v12  (1   17 )(0   20. )(.8   3.7 ))           (v23  (2   17 )(20.   43.8 )(3.7   10.4 ))
(v34  (3   17 )(43.8   68.6 )(10.4   20.9 ))         (v1  (0   17 )(0   0 )(0   .8 ))

(v11  (1   16 )(0   40. )(.8   3.7 ))           (v21  (2   15 )(40.   85.6 )(3.7   10.4 ))
(v32  (3   15 )(85.6   108.4 )(10.4   20.9 ))      (v43  (4   16 )(108.4   322.9
)(20.9   51.6 ))
```

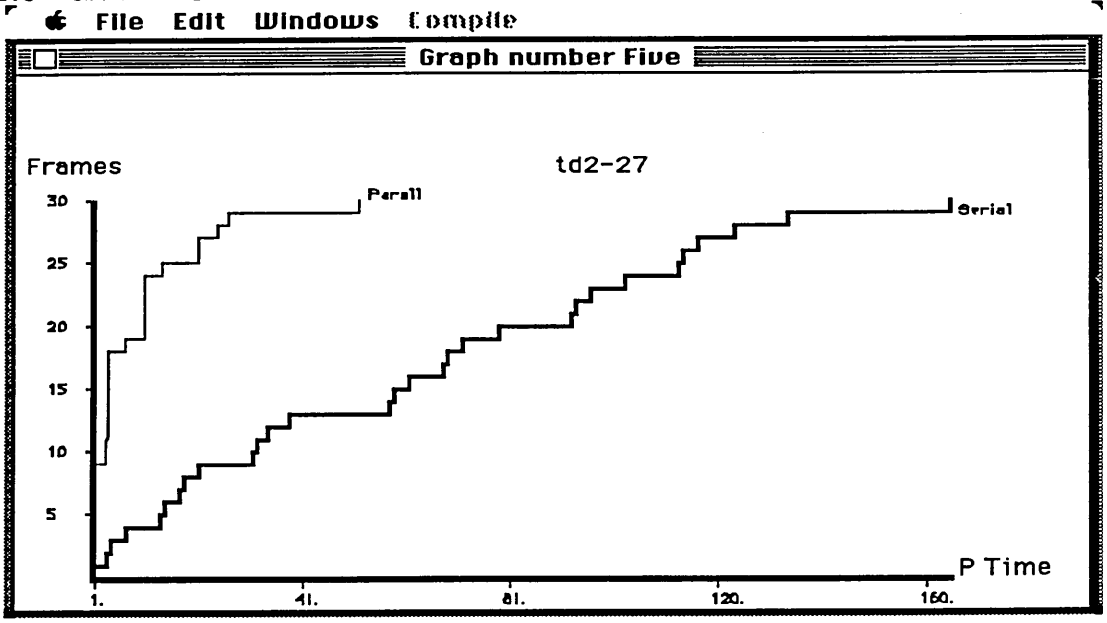
2.3 RUN-TIME GRAPH



2.4 GRAPH NUMBER 1/2



2.5 GRAPH NUMBER 5



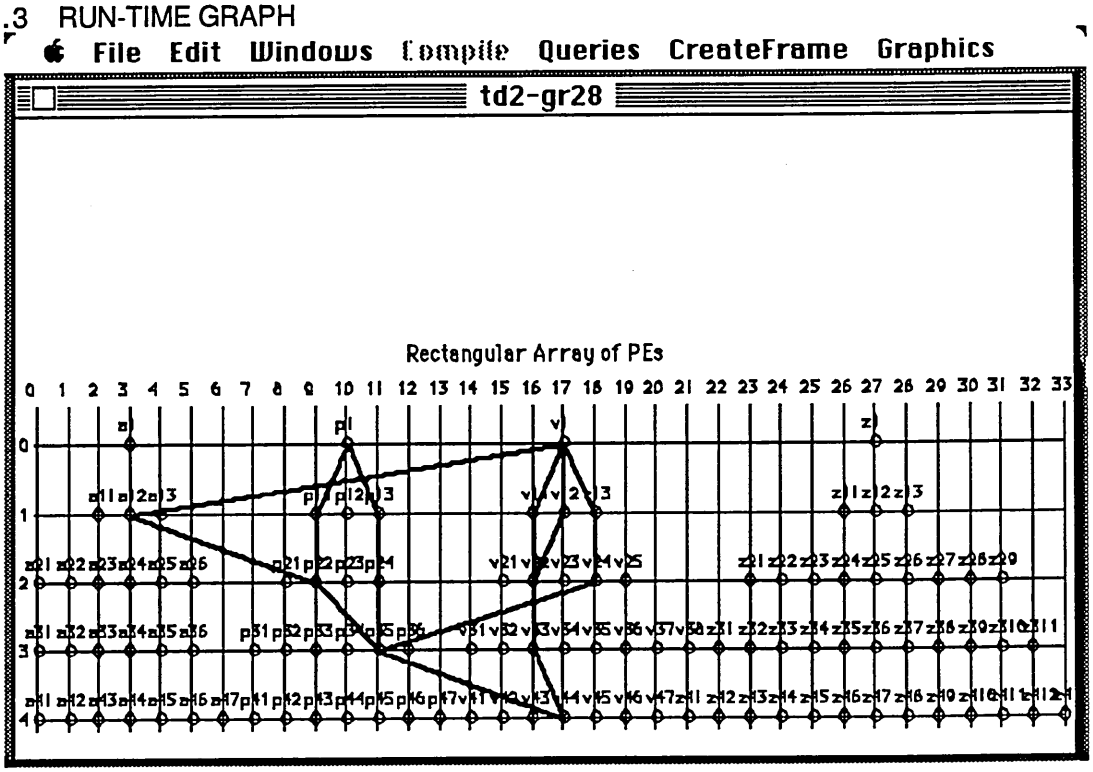
3.1 QUERY FRAME :

```
((P-hir  V-hir)        ;these are the names of hierarchies given by the user
  (?frame-name
                ((v13_inh_slot1  (value  ?V1))
                (p1_inh_slot1  (value  ?V2))
                (p2_inh_slot2  (value  ?V3))
                (p24_inh_slot1  (value  ?V4))
                (a1_inh_slot1  (value  ?V5))
                (v1_inh_slot1  (value  ?V6))  )))
```

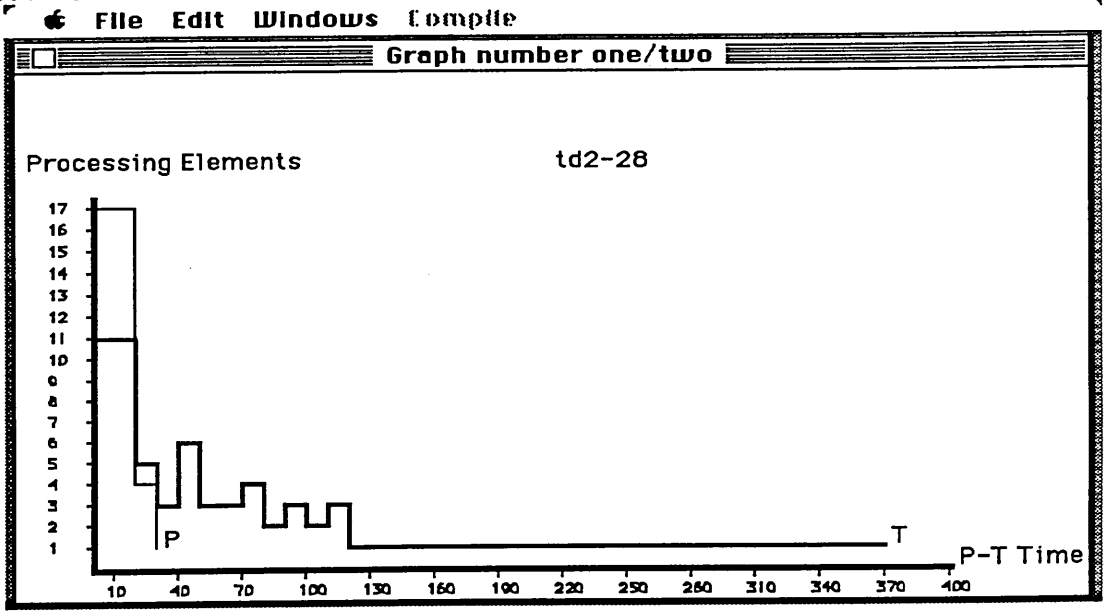
3.2 TIME-LIST :

```
(v1 (0   17 )(0   0 )(0   .8 ))                (v13 (1   18 )(0   39.6 )(.8   3.7 ))
(v24 (2   18 )(39.6   62.3 )(3.7   10.4 ))  (v1 (0   17 )(0   0 )(0   .8 ))
(p1 (0   10 )(0   0 )(0   .8 ))                (p11 (1   9 )(0   39.6 )(.8   3.1 ))
(p22 (2   9 )(39.6   60.4 )(3.1   7.8 ))     (p1 (0   10 )(0   0 )(0   .8 ))
(p13 (1   11 )(0   39.6 )(.8   3.1 ))                (p24 (2   11 )(39.6   60.4 )(3.1
7.4 ))
(p35 (3   11 )(60.4   250. )(7.4   19.9 ))  (v1 (0   17 )(0   0 )(0   .8 ))
(v12 (1   17 )(0   19.8 )(.8   3.1 ))                (v1 (0   17 )(0   0 )(0   .8 ))
(v11 (1   16 )(0   39.6 )(.8   3.1 ))                (v22 (2   16 )(39.6   81.2 )(3.1
7.5 ))
(v33 (3   16 )(81.2   107.3 )(7.5   13.4 ))       (v44 (4   17 )(107.3   375.4
)(13.4   30.4 ))
```

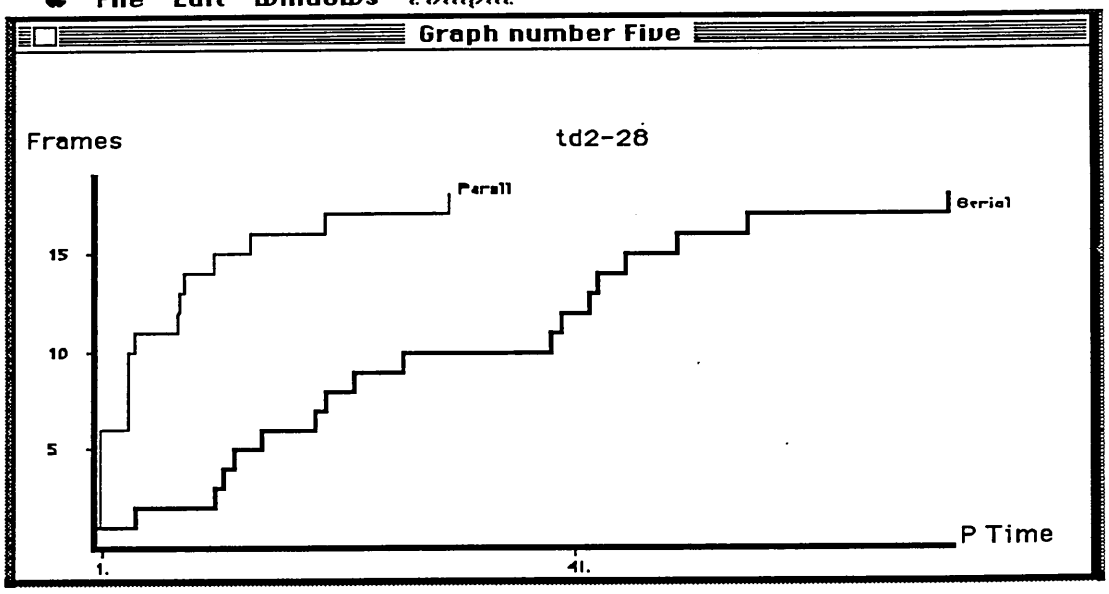
3.3 RUN-TIME GRAPH



3.4 GRAPH NUMBER 1/2



3.5 GRAPH NUMBER 5



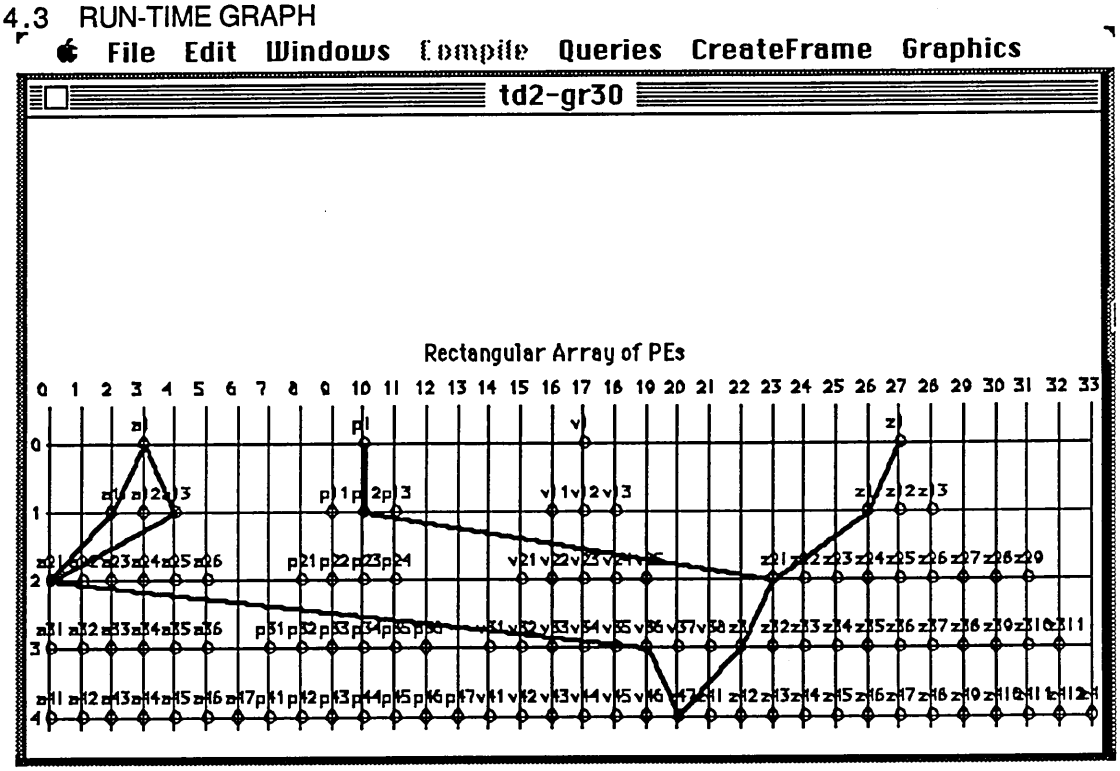
4.1 QUERY FRAME :

```
((P-hir   Z-hir   A-hir)        ;these are the names of hierarchies given by the user
  (?frame-name
              ((p1_inh_slot2  (value  ?V1))
                (z1_inh_slot1  (value  ?V2))
                (a1_inh_slot1  (value  ?V3))
                (a11_inh_slot1  (value  ?V4)) )))
```

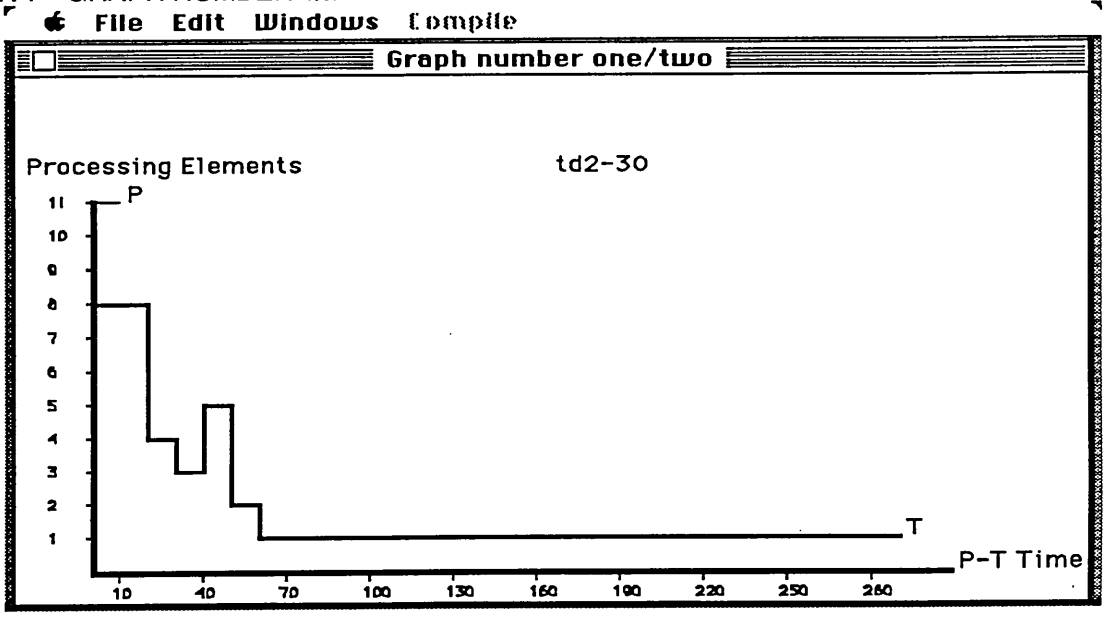
4.2 TIME-LIST :

```
(p1  (0   10 )(0   0 )(0   .8 ))
(p12  (1   10 )(0   16.2 )(.8   3.1 ))
(z1  (0   27 )(0   0 )(0   .8 ))
(z11  (1   26 )(0   32.4 )(.8   3.1 ))
(z21  (2   23 )(32.4   259.2 )(3.1   8.7 ))
(z31  (3   22 )(259.2   298.8 )(8.7   17.6 ))
(a1  (0   3 )(0   0 )(0   .7 ))
(a13  (1   4 )(0   31.6 )(.7   2.9 ))
(a1  (0   3 )(0   0 )(0   .7 ))
(a11  (1   2 )(0   31.6 )(.7   3.5 ))
(a21  (2   0 )(31.6   113.1 )(3.5   10.1 ))
```

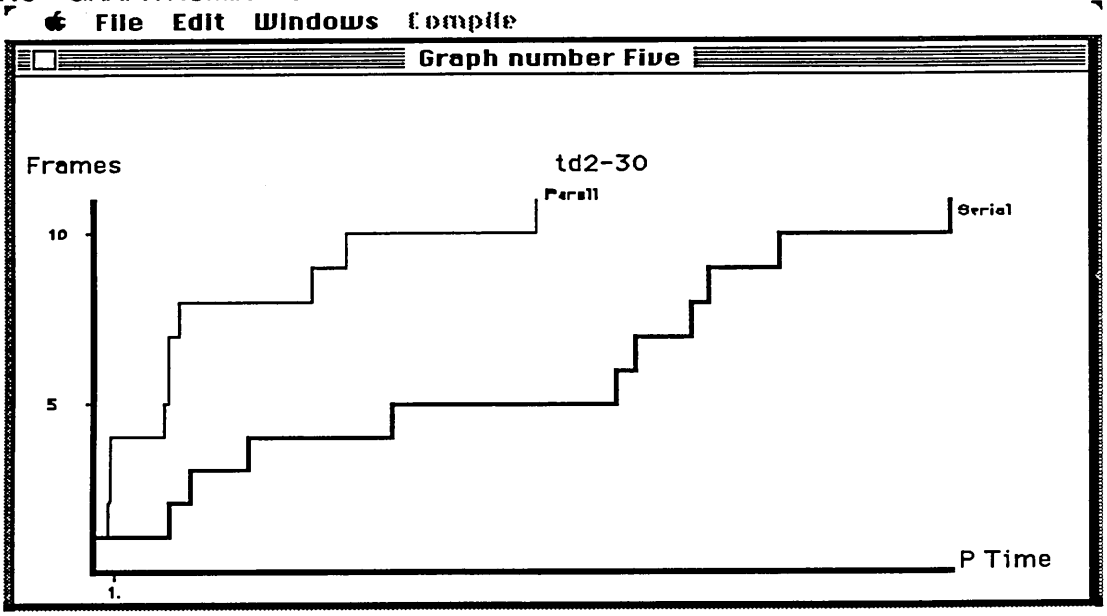
4.3 RUN-TIME GRAPH



4.4 GRAPH NUMBER 1/2



4.5 GRAPH NUMBER 5



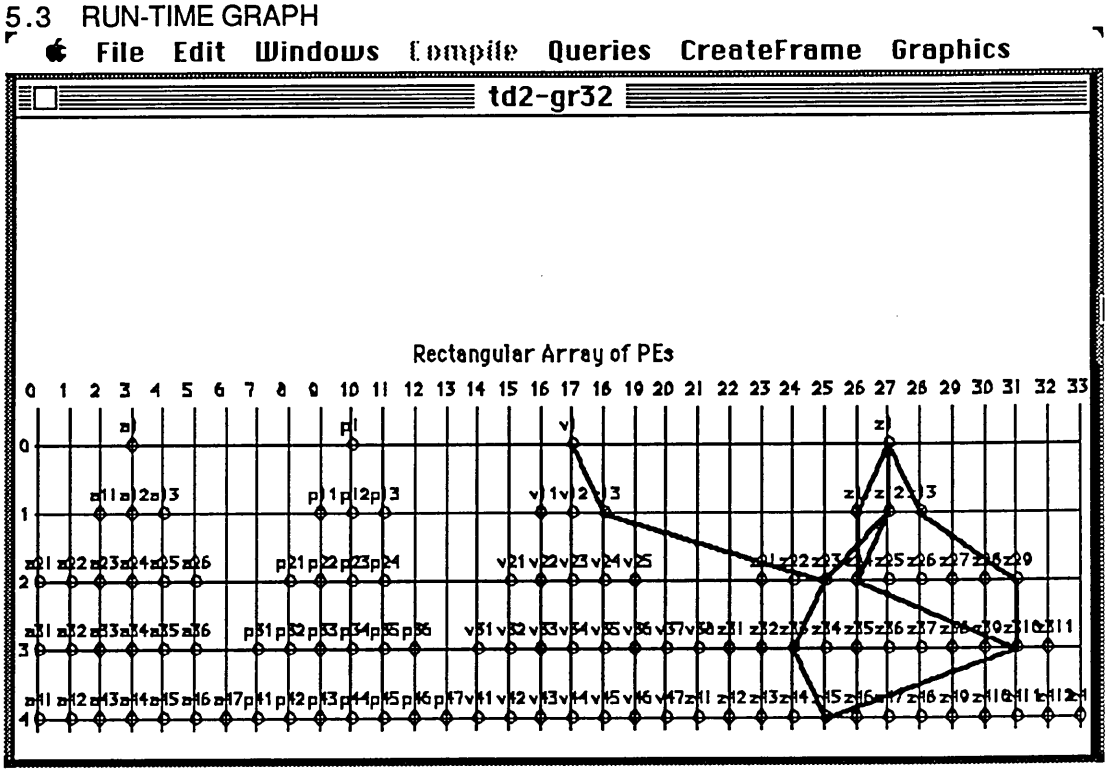
5.1 QUERY FRAME :

```
((Z-hir   V-hir)        ;these are the names of hierarchies given by the user
  (?frame-name
                ((z13_inh_slot1  (value  ?V1))
                  (v13_inh_slot1  (value  ?V2))
                (z1_inh_slot1  (value  ?V3))  )))
```

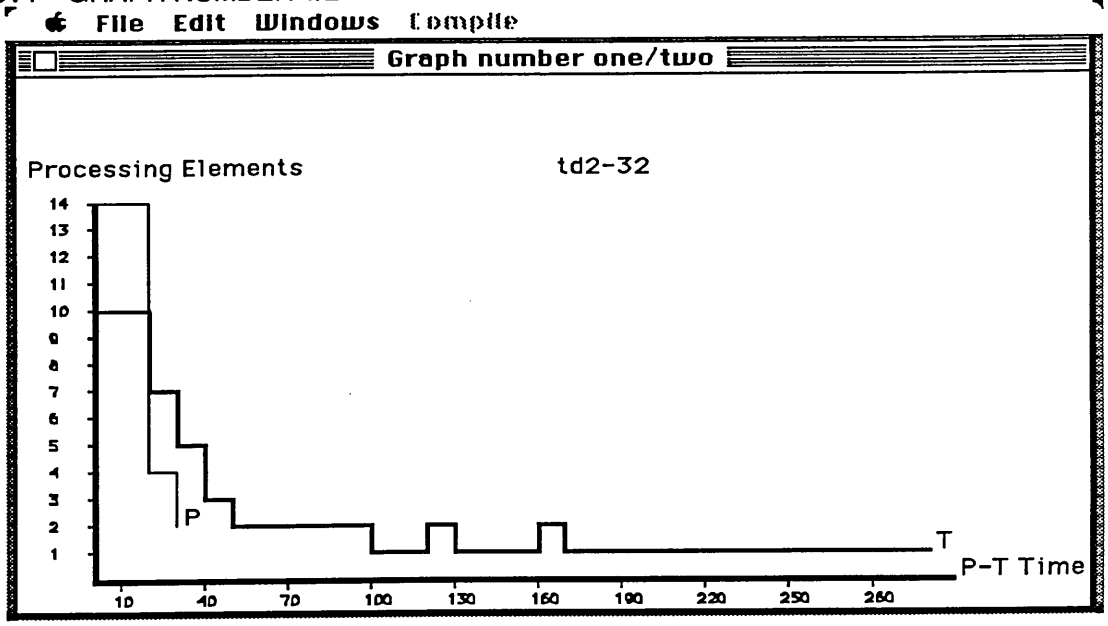
5.2 TIME-LIST :

```
(z1 (0   27 )(0   0 )(0   .8 ))
(z11 (1   26 )(0   26. )(.8   3.1 ))
(z1 (0   27 )(0   0 )(0   .8 ))
(z12 (1   27 )(0   13. )(.8   3.1 ))
(z24 (2   26 )(13.   39. )(3.1   6.9 ))
(z1 (0   27 )(0   0 )(0   .8 ))
(z13 (1   28 )(0   26. )(.8   3.7 ))
(z29 (2   31 )(26.   89.2 )(3.7   10.4 ))
(z310 (3   31 )(89.2   182.8 )(10.4   20.9 ))
(v1 (0   17 )(0   0 )(0   0 ))
(v13 (1   18 )(0   20.8 )(0   .9 ))
(z1 (0   27 )(0   0 )(0   .8 ))
(z12 (1   27 )(0   13. )(.8   3.1 ))
(z23 (2   25 )(13.   118.6 )(3.1   9. ))
(z33 (3   24 )(118.6   154.2 )(9.   18.5 ))
(z45 (4   25 )(154.2   308.9 )(18.5   34.9 ))
```

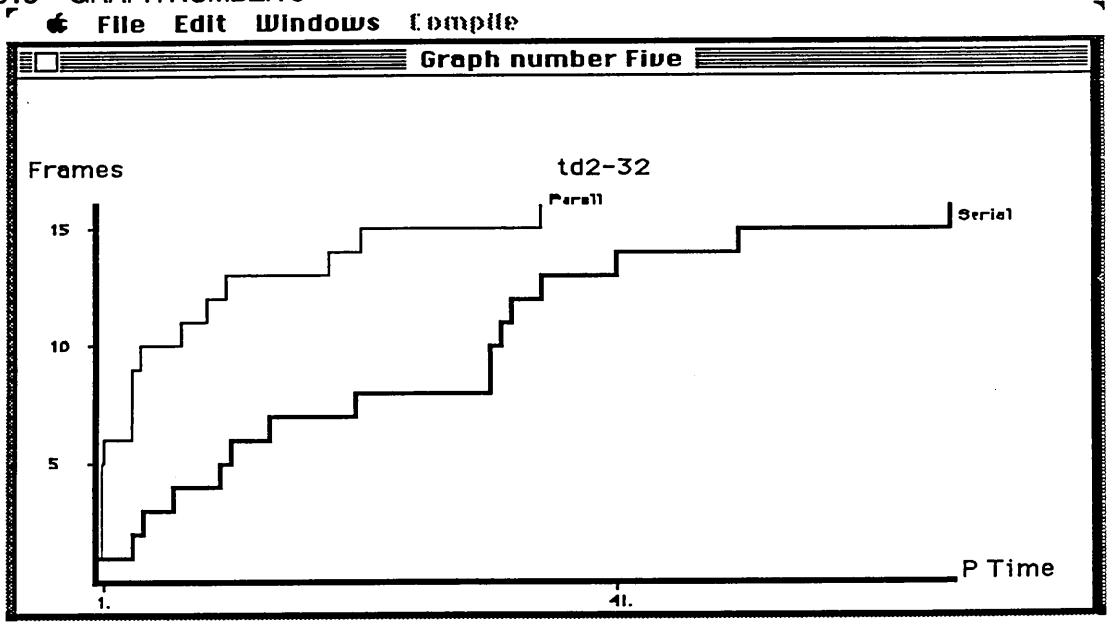
5.3 RUN-TIME GRAPH



5.4 GRAPH NUMBER 1/2



5.5 GRAPH NUMBER 5



In this type of query, similar to query-object number two, and three, at each interrogation a query-frame is created. The query-frame consists of a conjunction of two or more frames with some shared slot-names and possible values. The user may provide one or more frame-names. The missing slots/values are represented by unbound variables, which will be bound to their appropriate values at run time.

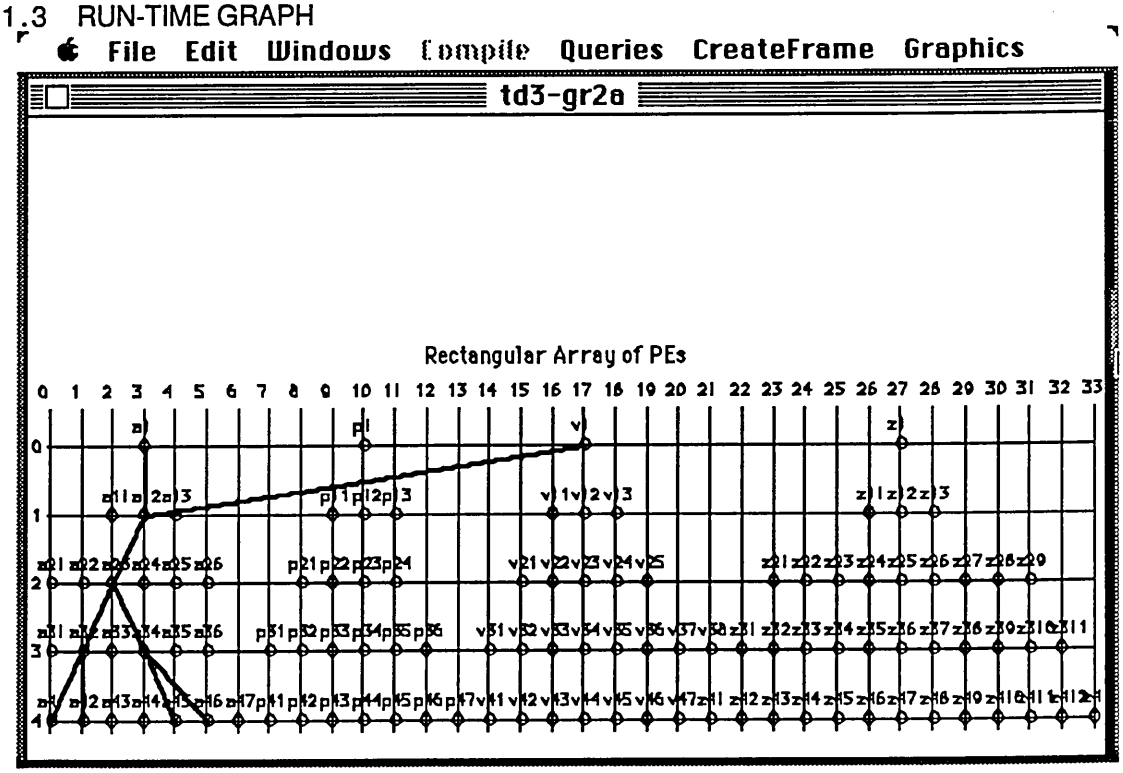
1.1 QUERY FRAME:

```
((a23  a32  a34) ; frame-names
        ((a32_inh_slot1  (value  ?v1))
        (?slot1  (value  (a32_inh_v8  a41_inh_v8)))
        (v1_inh_slot1(value(v1_inh_v1  a12_inh_v5  a23_inh_v6)))  ))
```

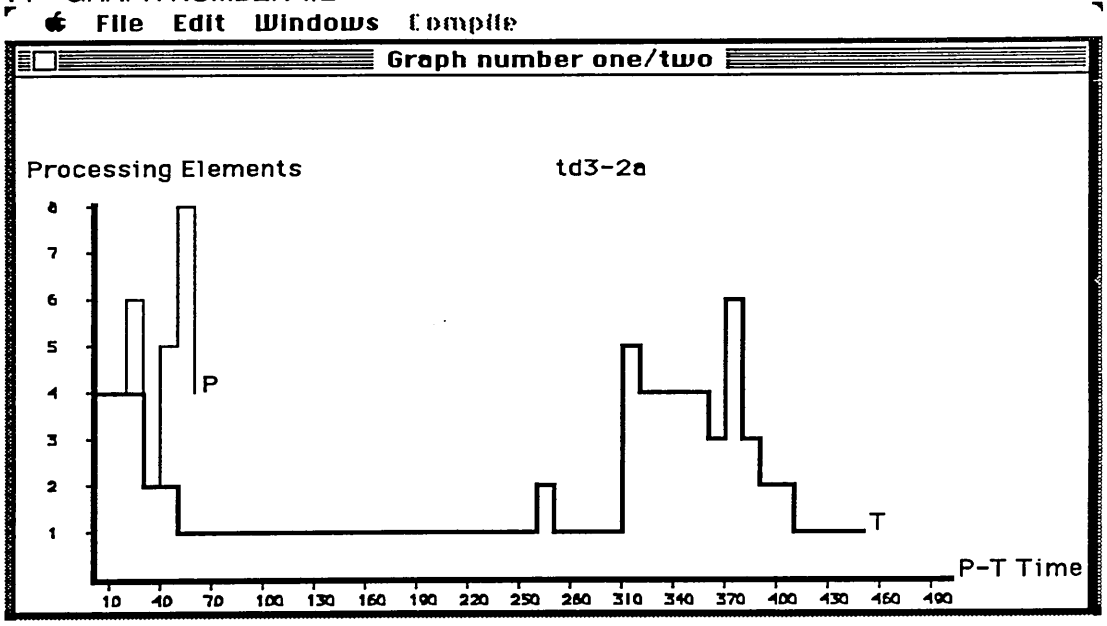
1.2 TIME-LIST :

```
(a41 (4   0 )(0   11.3 )(0   14.1 ))          (v1 (0   17 )(11.3   11.3 )(14.1   14.9 ))
(a1 (0   3 )(11.3   11.3 )(14.1   14.1 ))    (a12 (1   3 )(11.3   296.3 )(14.1   16. ))
(a23 (2   2 )(296.3   341.9 )(16.   21.5 ))        (v1 (0   17 )(7.4   7.4 )(5.3   6.1
) )
(a1 (0   3 )(7.4   7.4 )(5.3   6.7 ))          (a12 (1   3 )(7.4   259.4 )(6.7   14.6 ))
(a23 (2   2 )(259.4   305.8 )(14.6   30.4 ))       (a32 (3   1 )(305.8   361.8
)(30.4   49.6 ))
(a41 (4   0 )(361.8   423.8 )(49.6   69.4 ))       (a32 (3   1 )(305.8   361.8
)(30.4   49.6 ))
(a42 (4   1 )(361.8   392.8 )(49.6   69.7 ))       (a34 (3   3 )(305.8   361.8
)(30.4   49.3 ))
(a45 (4   4 )(361.8   421.8 )(49.3   69.1 ))       (a34 (3   3 )(305.8   361.8
)(30.4   49.3 ))
(a46 (4   5 )(361.8   451.8 )(49.3   69.1 ))
```

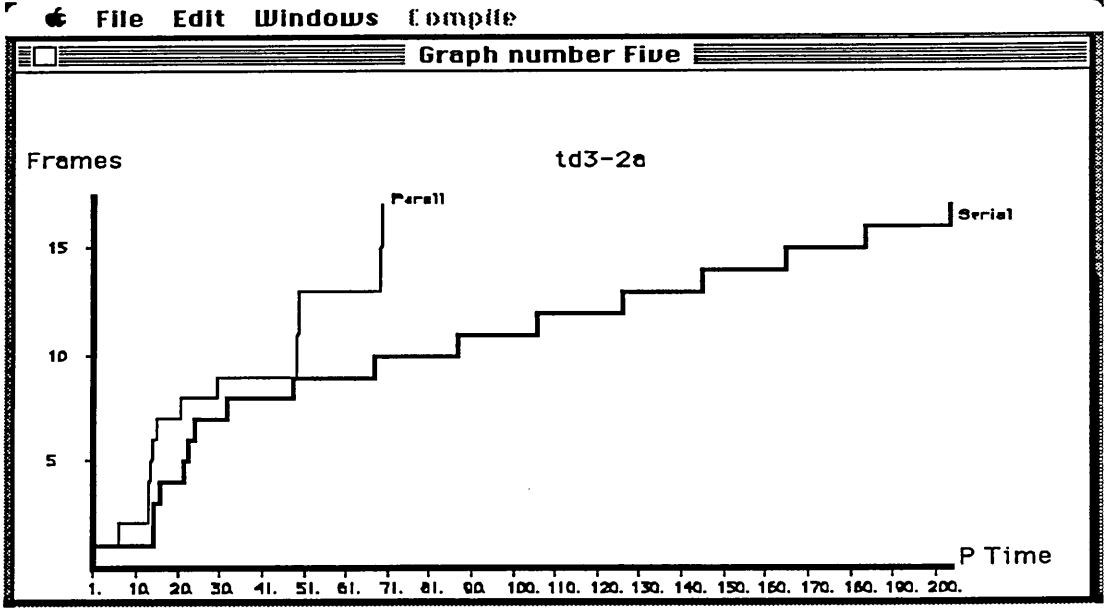
1.3 RUN-TIME GRAPH



1.4 GRAPH NUMBER 1/2



1.5 GRAPH NUMBER 5



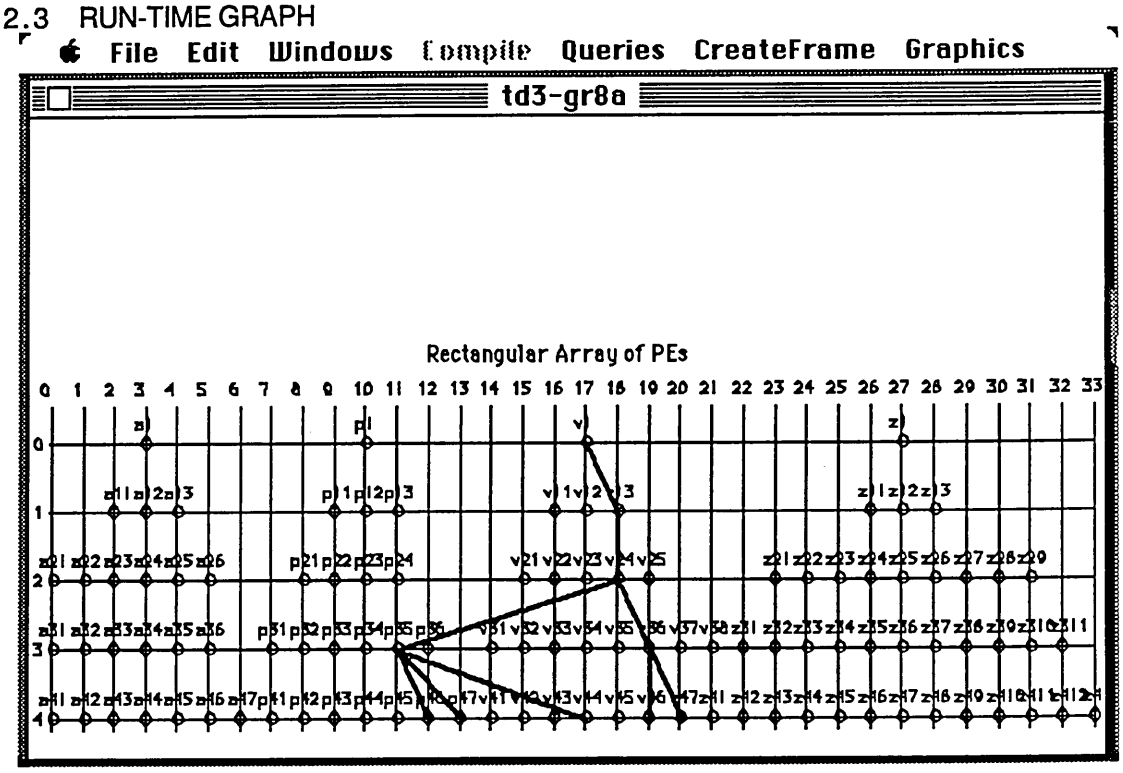
2.1 QUERY FRAME :

```
((v24    ?frame-names)
      ((v1_inh_slot1  (value  ?V1))
      (v13_inh_slot1  (value  ?V2))
      (v24_inh_slot1  (value  ?V3))  ))
```

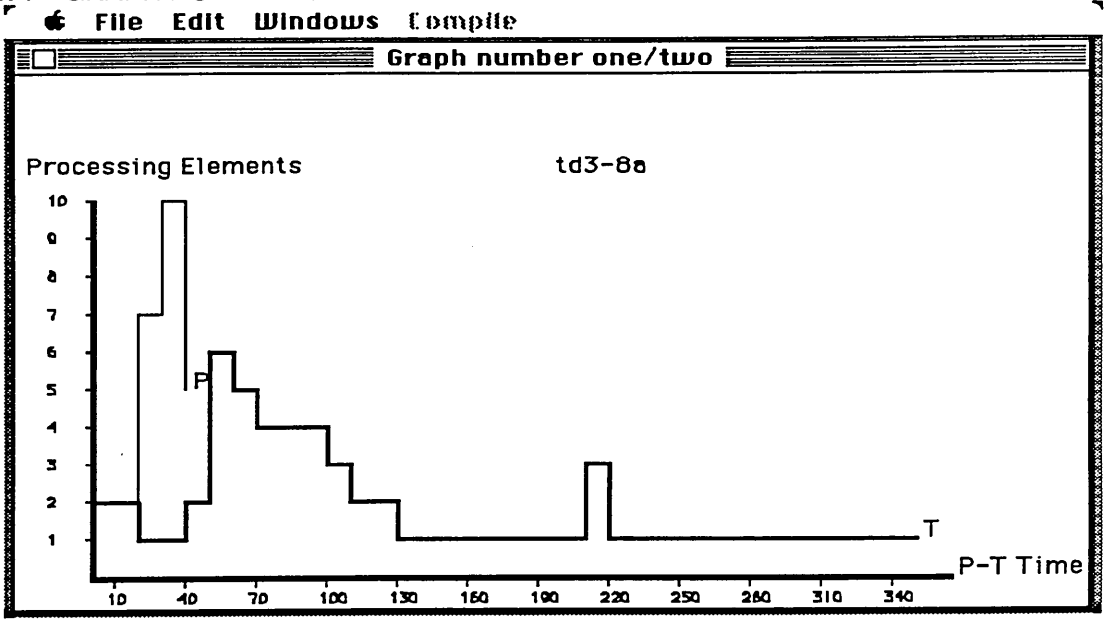
2.2 TIME-LIST :

```
(v1  (0   17  )(5.3   5.3  )(7.   7.8  ))
(v13  (1   18  )(5.3   31.5  )(7.8   10.7  ))
(v24  (2   18  )(31.5   47.5  )(10.7   18.4  ))
(v36  (3   19  )(47.5   87.1  )(18.4   29.3  ))
(v46  (4   19  )(87.1   106.9  )(29.3   40.2  ))
(v36  (3   19  )(47.5   87.1  )(18.4   29.3  ))
(v47  (4   20  )(87.1   126.7  )(29.3   40.2  ))
(p35  (3   11  )(47.5   205.9  )(18.4   29.6  ))
(p46  (4   12  )(205.9   247.5  )(29.6   43.2  ))
(p35  (3   11  )(47.5   205.9  )(18.4   29.6  ))
(p47  (4   13  )(205.9   268.3  )(29.6   43.2  ))
(p35  (3   11  )(47.5   205.9  )(18.4   29.6  ))
(v44  (4   17  )(205.9   351.5  )(29.6   39.6  ))
```

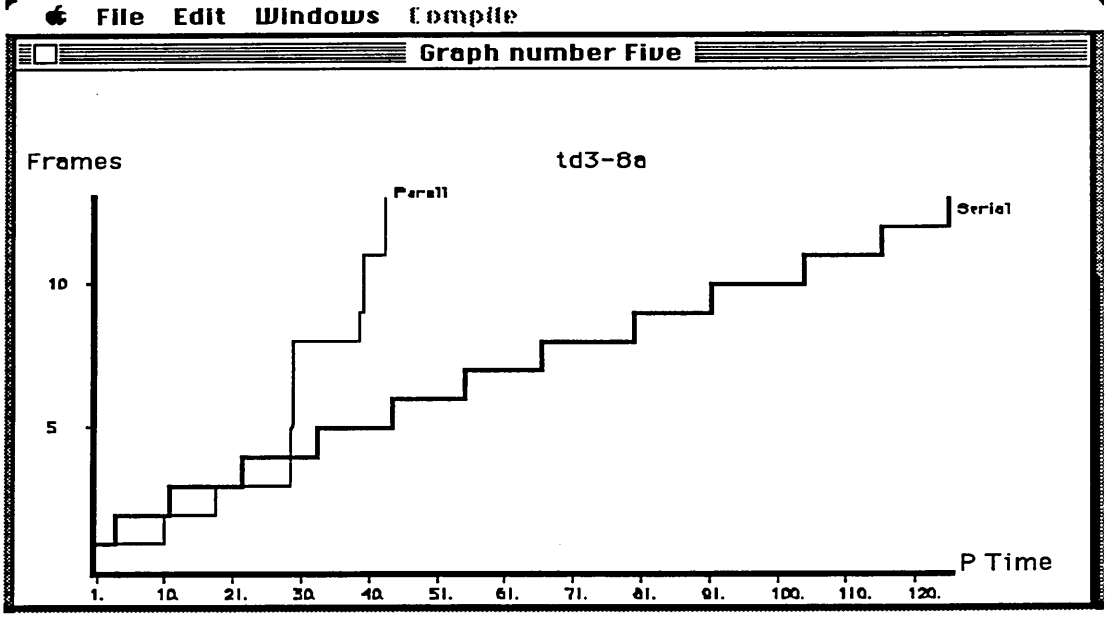
2.3 RUN-TIME GRAPH



2.4 GRAPH NUMBER 1/2



2.5 GRAPH NUMBER 5



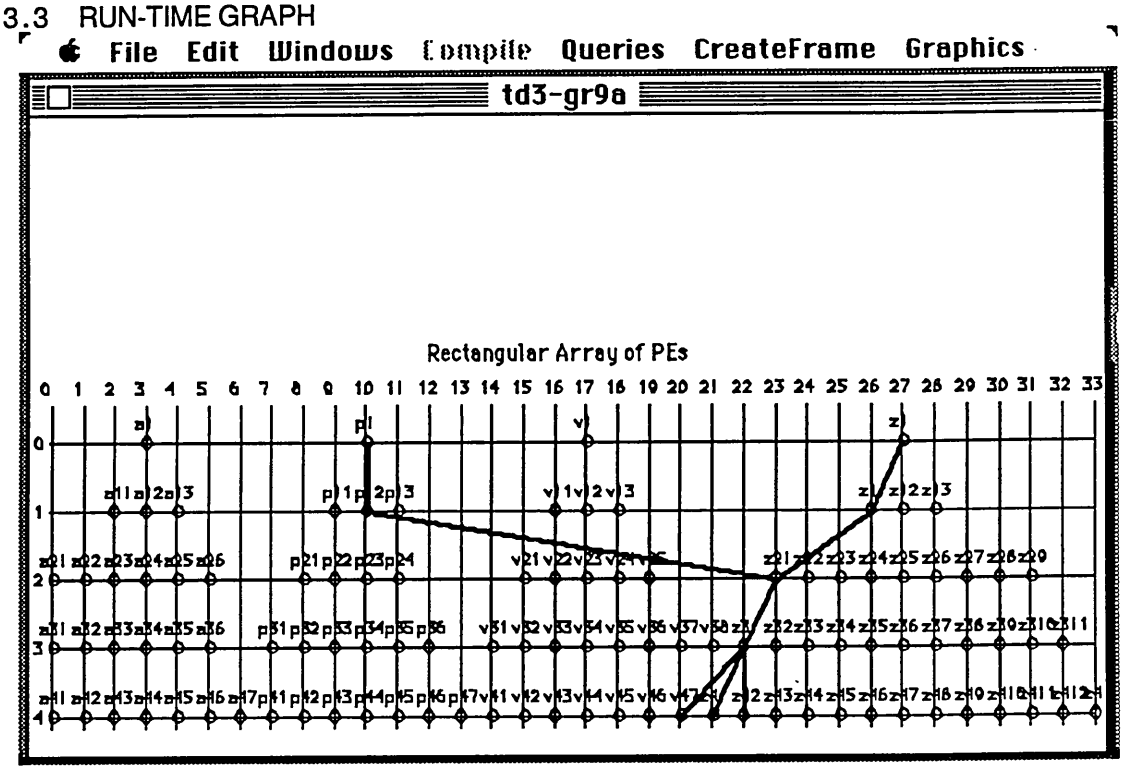
3.1 QUERY FRAME :

```
((z21   v48   ?frame-names)
        ((z1_inh_slot1  (value  ?V1))
        (p1_inh_slot1  (value  ?V2))
        (p1_inh_slot2  (value  ?V3))
        (z21_inh_slot1  (value  ?V4))  ))
```

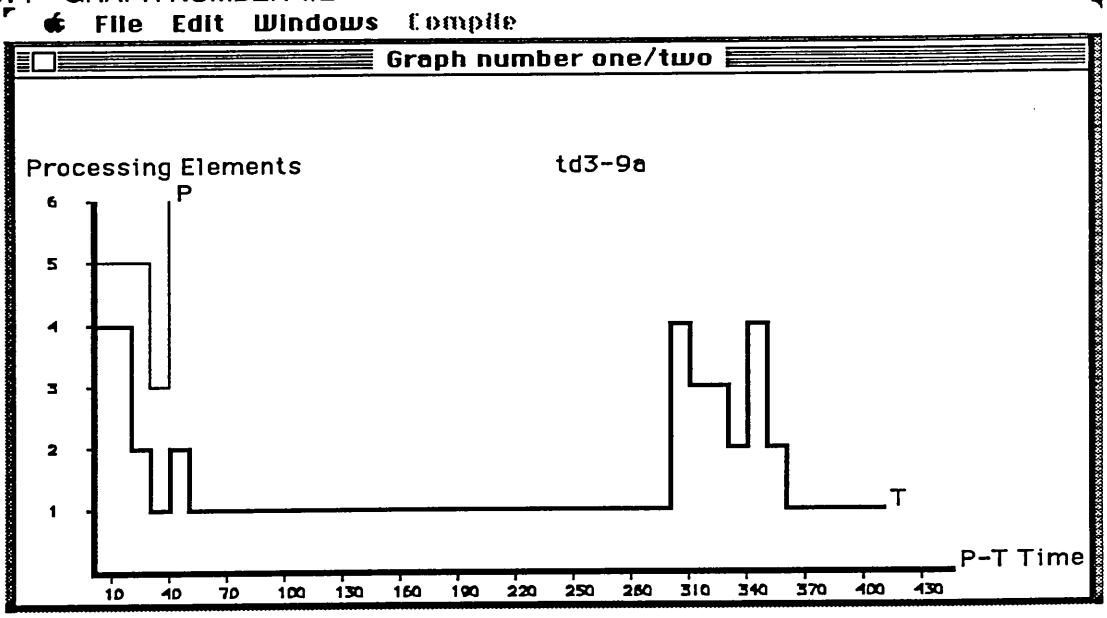
3.2 TIME-LIST :

```
(p1  (0   10 )(6.9   6.9 )(5.8   7.4 ))
(p12  (1   10 )(6.9   24.1 )(7.4   13.4 ))
(z1  (0   27 )(6.9   6.9 )(5.8   6.6 ))
(z11  (1   26 )(6.9   36.1 )(6.6   8.9 ))
(z21  (2   23 )(36.1   290.9 )(8.9   19. ))
(z31  (3   22 )(290.9   338.3 )(19.   32.5 ))
(z41  (4   21 )(338.3   387.7 )(32.5   46.3 ))
(z31  (3   22 )(290.9   338.3 )(19.   32.5 ))
(z42  (4   22 )(338.3   363. )(32.5   45.1 ))
(z31  (3   22 )(290.9   338.3 )(19.   32.5 ))
(v47  (4   20 )(338.3   412.4 )(32.5   46.6 ))
```

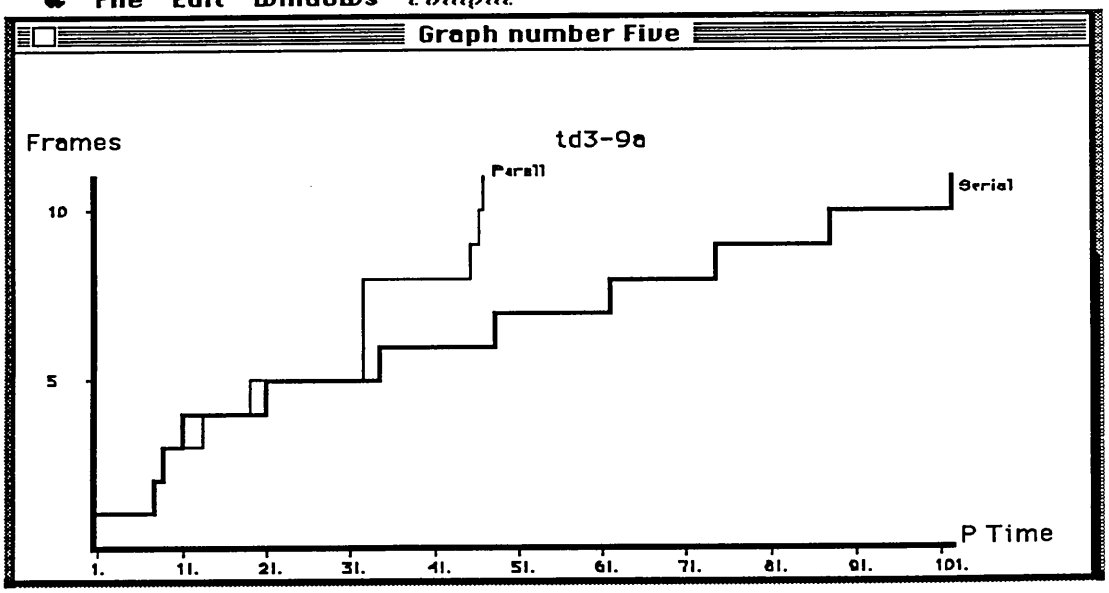
3.3 RUN-TIME GRAPH



3.4 GRAPH NUMBER 1/2



3.5 GRAPH NUMBER 5



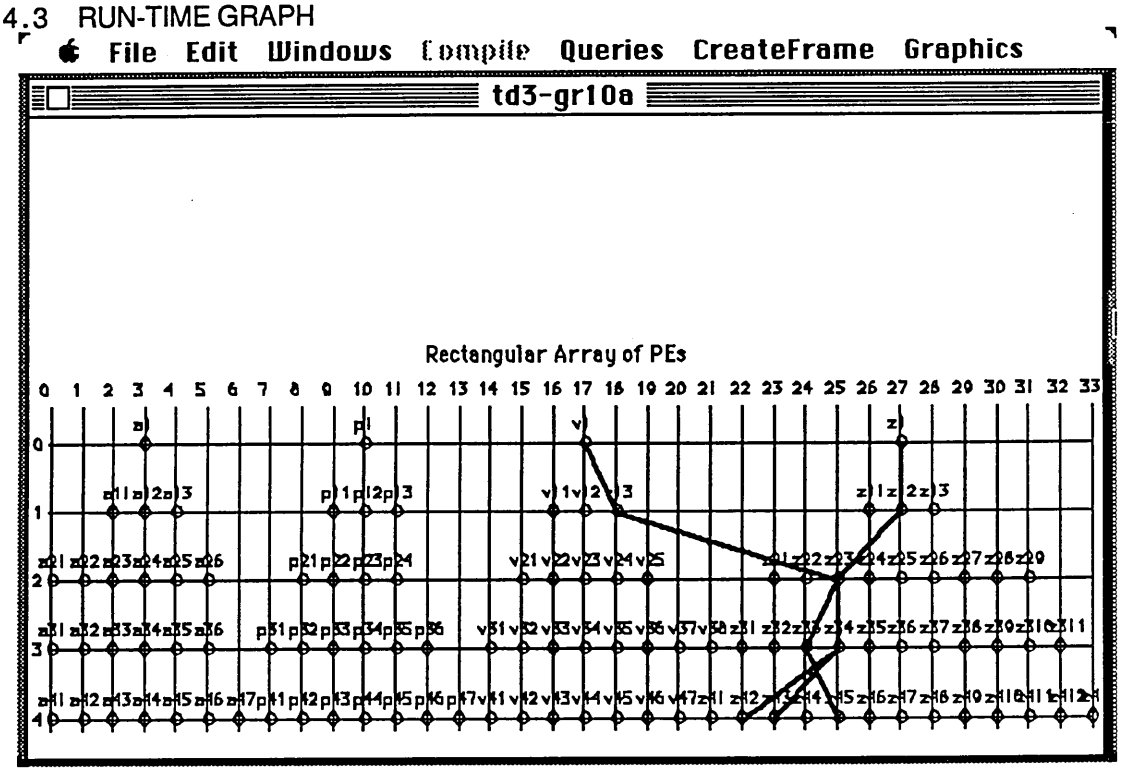
4.1 QUERY FRAME :

```
((z23   z44   ?frame-names)
        ((z1_inh_slot1  (value  ?V1))
        (v1_inh_slot1  (value  ?V2))
        (?slot1  (value  v13_inh_v1))
        (z23_inh_slot1  (value  ?V3))  ))
```

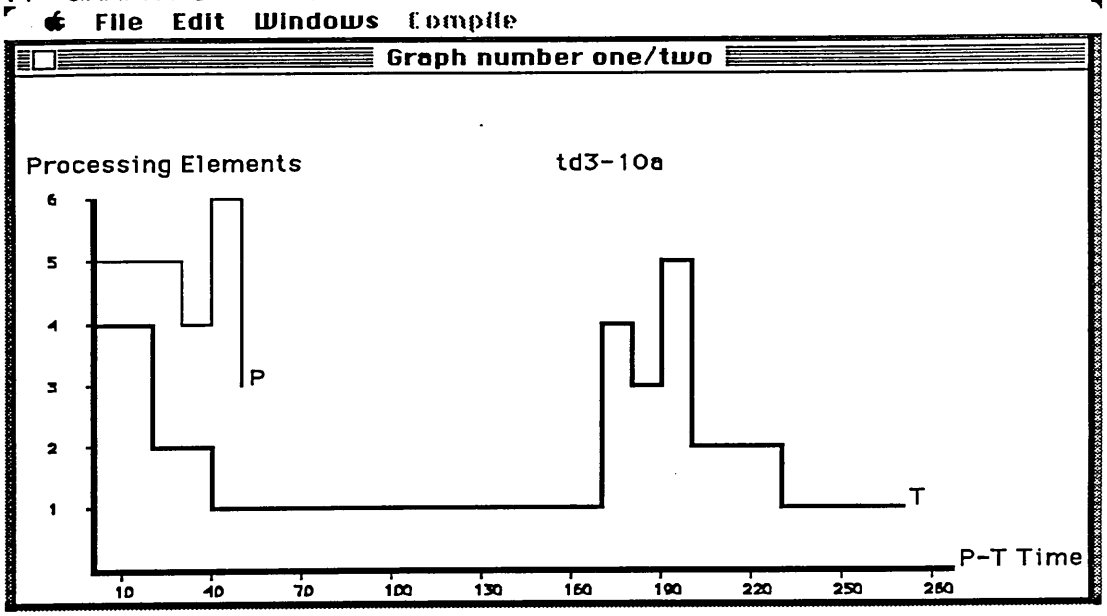
4.2 TIME-LIST :

```
(v1  (0   17 )(7.   7. )(13.6   14.4 ))
(v13  (1   18 )(7.   36.4 )(14.4   17.3 ))
(z1  (0   27 )(7.   7. )(13.6   14.4 ))
(z12  (1   27 )(7.   21.7 )(14.4   16.7 ))
(z23  (2   25 )(21.7   162.5 )(16.7   26.3 ))
(z33  (3   24 )(162.5   212.5 )(26.3   38.1 ))
(z45  (4   25 )(212.5   264.5 )(38.1   53.2 ))
(z34  (3   25 )(162.5   187.5 )(26.3   38.1 ))
(z42  (4   22 )(187.5   291.5 )(38.1   50.2 ))
(z34  (3   25 )(162.5   187.5 )(26.3   38.1 ))
(z43  (4   23 )(187.5   265.5 )(38.1   53.5 ))
```

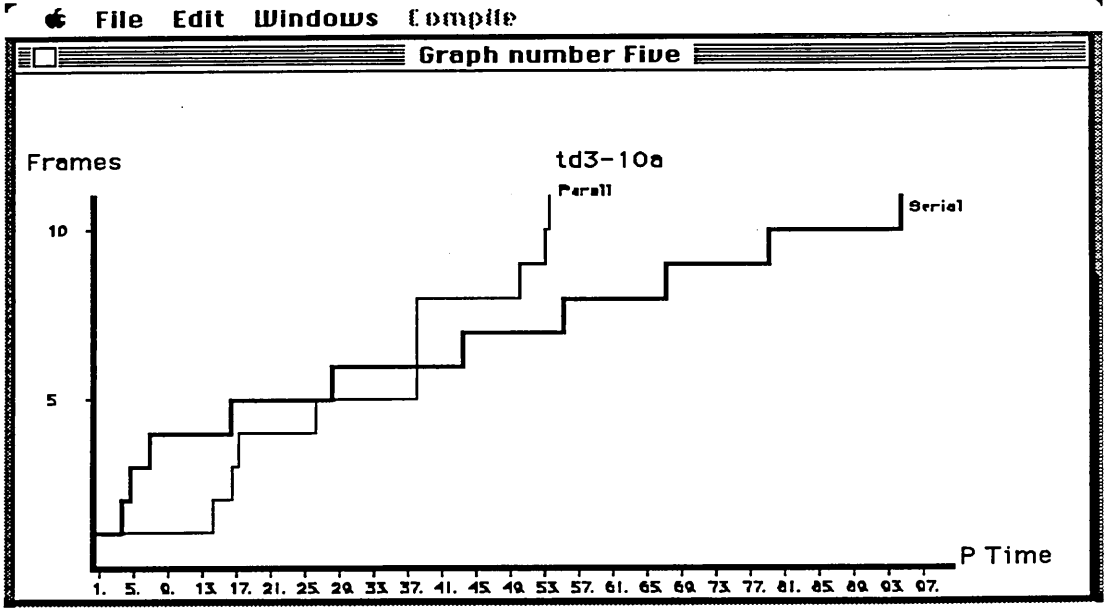
4.3 RUN-TIME GRAPH



4.4 GRAPH NUMBER 1/2



4.5 GRAPH NUMBER 5



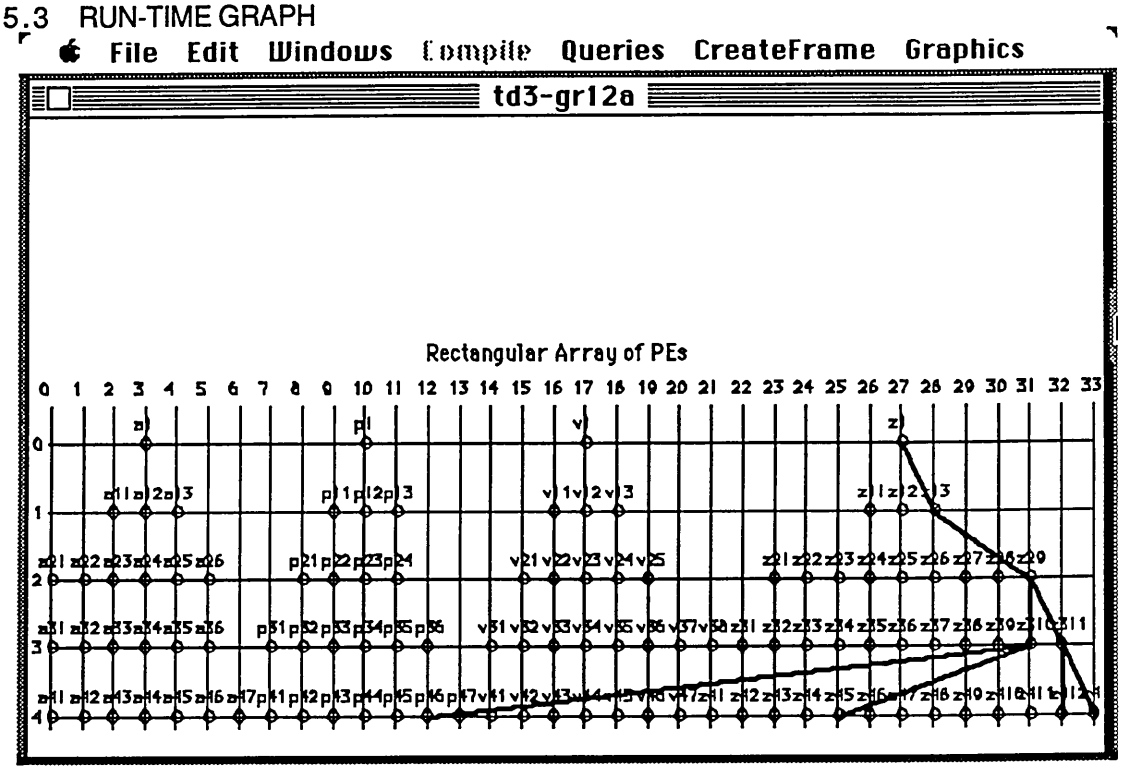
5.1 QUERY FRAME :

```
((z29    ?frame-names)
         ((z1_inh_slot1  (value  ?V1))
        (z13_inh_slot1  (value  ?V2))
        (z29_inh_slot1  (value  ?V3))  ))
```

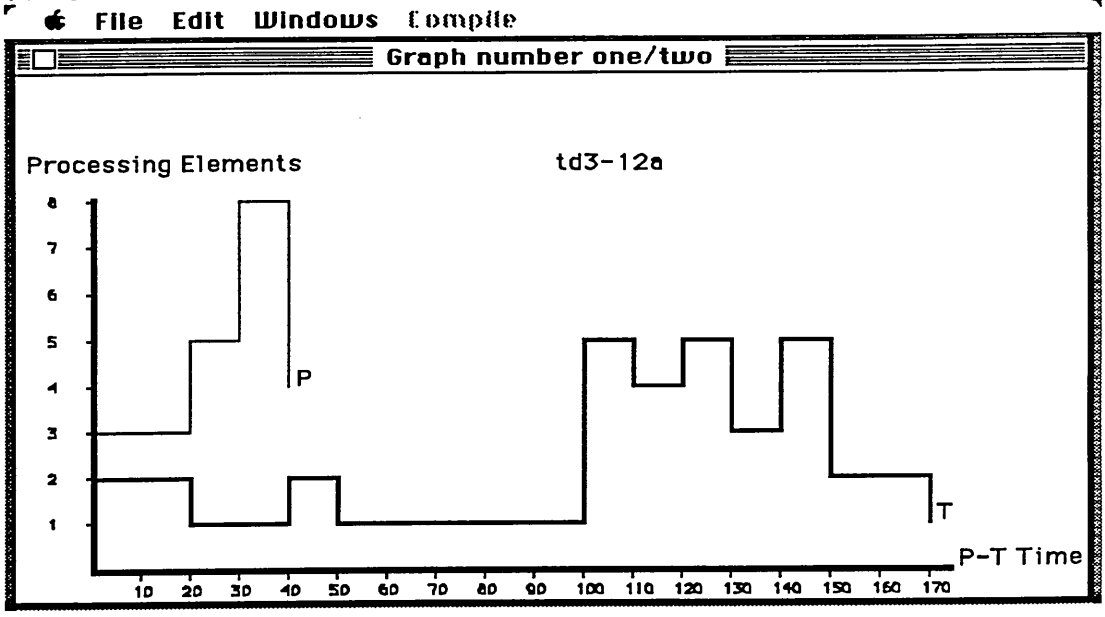
5.2 TIME-LIST :

```
(z1  (0   27 )(5.3   5.3 )(3.5   4.3 ))
(z13  (1   28 )(5.3   31.5 )(4.3   7.2 ))
(z29  (2   31 )(31.5   95.1 )(7.2   14.9 ))
(z310  (3   31 )(95.1   113.8 )(14.9   27.3 ))
(z45  (4   25 )(113.8   252.4 )(27.3   39.7 ))
(z310  (3   31 )(95.1   113.8 )(14.9   27.3 ))
(p46  (4   12 )(113.8   509.8 )(27.3   40.6 ))
(z311  (3   32 )(95.1   132.5 )(14.9   27. ))
(z412  (4   32 )(132.5   151.2 )(27.   39.1 ))
(z311  (3   32 )(95.1   132.5 )(14.9   27. ))
(z413  (4   33 )(132.5   169.9 )(27.   39.4 ))
```

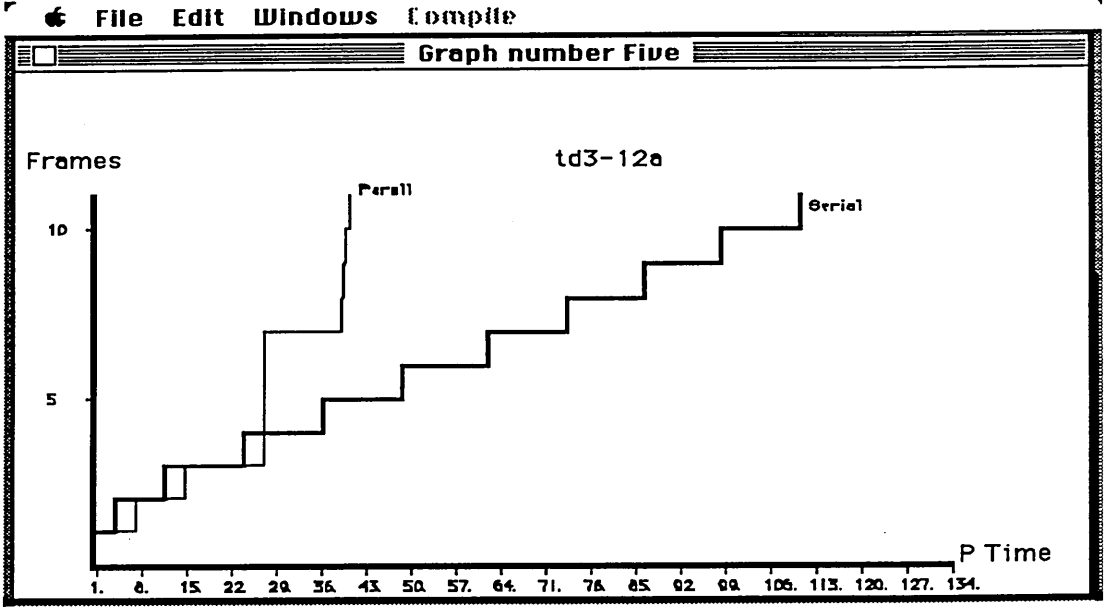
5.3 RUN-TIME GRAPH



5.4 GRAPH NUMBER 1/2



5.5 GRAPH NUMBER 5



APPENDIX E : JOIN IN FRAMES

There are several ways to implement the join operator as a higher level query in the SM. A description of two options is set out below. But before discussing these options, the way in which frames are concatenated, as with the concatenation of tuples of two relation in join, is examined. In figure 1, a comparison is made between two hierarchies and their respective relations, where tuples are assumed to be similar to frames. As mentioned in section 4.73, chapter 4, all the examples are simplified for a clearer demonstration of the smililarities between hierarchies and relations.

To join the relations in figure 1 on their common attribute, a SQL-like query can be made as follows:
JOIN first-relation with second-relation where
comm-att of the first-relation = comm-att of the second-relation

The operation would involve the concatenation of each tuple of the first relation with every tuple of the second relation, where the value of the common attribute of the first relation is equal to the value of the common attribute of the second relation.

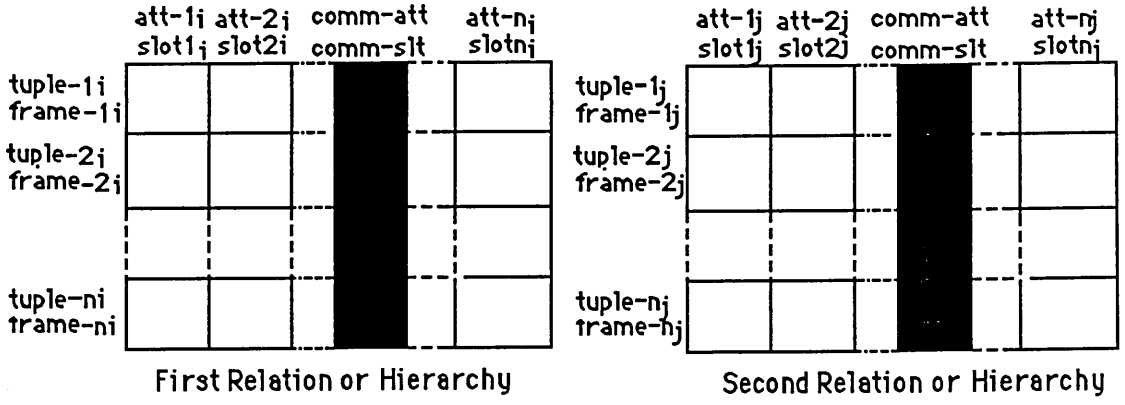


Figure 1.

The resultant table would then contain all the successful concatenations. The number of columns of the resultant table is equal to the number of columns of the first relation plus the number of columns of the second relation (ie, 1i-ni + 1j-nj). The number of rows would be equal to the number of concatenation of all the tuples in the first relation with all those tuples in the second relations that have satisfied the given condition (in this example the condition is: first-relation comm-att = second-relation comm-att). In section 4.73, chapter 4, a simple example of join on two relations is given showing the resultant relation with increased numbers of columns and rows.

By drawing an analogy between frames and tuples, we can similarly join two hierarchies where the values of the common-slots in the first-hierarchy are equal to the values of common-slots in the second hierarchy. Here, it must be noted that in the SM it will only be possible to join hierarchies after identifying all the appropriate frames in them. The full version[1] of these frames can then be concatenated according to the given conditions. The result of concatenation may be as follows (see figure 1):

```
( (frame-1i   (slot-1i (value v1i))....(common-slot (value v))....  (slotni (value vni))
   frame-1j   (slot-1j (value v1j))....(common-slot (value v))....  (slotnj (value vnj))  )
  .
  .
  (frame-1i   (slot-1i (value v1i)) ....(common-slot (value v))....  (slotni (value vni))
   frame-nj   (slot-nj (value vnj))....(common-slot (value v))....  (slotnj (value vnj))  )

  (frame-2i   (slot-2i (value v2i)) ....(common-slot (value v))....  (slotni (value vni))
    frame-1j  (slot-1j (value v1j))....(common-slot (value v))....  (slotnj (value vnj))  )
  .
  .
  (frame-2i   (slot-2i (value v2i)) ....(common-slot (value v))....  (slotni (value vni))
   frame-nj   (slot-nj (value vnj))....(common-slot (value v))....  (slotnj (value vnj))  )
  .
  .
  (frame-ni   (slot-ni (value vni)) ....(common-slot (value v))....  (slotni (value vni))
    frame-1j  (slot-1j (value v1j))....(common-slot (value v))....  (slotnj (value vnj))  )
  .
  .
  (frame-ni   (slot-ni (value vni)) ....(common-slot (value v))....  (slotni (value vni))
    frame-nj  (slot-nj (value vnj))....(common-slot (value v))....  (slotnj (value vnj))  ) )
```

Furthermore, in joining two hierarchies it is possible that instead of only one group of frames (as above), several groups are returned to the controller. In each group all the frames have the same value for the common slot-name, but the values for the common slots in each group may be different.

First option:

The first option involves a 2 stage propagation. In the first stage we can ask for the full version of every frame in the first hierarchy that contain the common slot-

---

[1] The full version of a frame is a frame, which has undergone all the necessary operations, including inheritance.

names. Eg a query can be made based on the hierarchies given in figures 4.16 and 4.17, chapter 4 :

```
( (hierarchies 'A' )
  (?frame-names ;this is the Employee's name
  (City-name (value ?V1)) ));the only common slot-name in this example
```

When the first stage is completed, several groups of frames will be returned to the controller, where in each group the common slot-name has the same value in each frame. In the example given above, after the first propagation the following groups of frames will be returned to the controller:

```
((Frame-name : Terry      ((Frame-name : Dary ((Frame-name : Ian
  (Emp-no : E107)           (Emp-no : E912)      (Emp-no : E239)
  (City-name : London))     (City-name : Leeds))  (City-name : Leeds))
```

There are only two groups in this example, one of which contains only one frame. In a more complex example, we could have received one group that share the common slot-name with Leeds, another group with London and another group with Sheffield, as their respective values.

In the second stage, we can then ask for every group of frames in the second hierarchy 'B', that contain the City-name with each different value obtained from the first stage (eg, Leeds, London, Sheffield etc). The controller would then receive the result of the second stage consisting of several groups of frames. That is:

```
( (hierarchies 'B')
  (?frame-names
  (Or    (City-name (value London))
         (City-name (value Leeds)) )  ) )
```

The result from the second propagation is returned to the controller:

```
((Frame-name : W34       ((Frame-name : W92
  (City-name : Leeds))      (City-name : Leeds))
```

At this stage, the controller has groups of frames from both stages.

As mentioned above, each group may be viewed as a relation, which implies we can now join one relation from the first stage with another relation in the second stage that share the same value for the common slot-name.

Note that it may be possible to perform the concatenation in the second stage at the array level, rather than at the controller level. That is, instead of sending all the appropriate frames obtained in the first stage to the controller, they can be propagated through out the second hierarchy in the second stage. This can be done by each packet

which, in addition to the existing information that it contains, carries all the relevant frames found in the first stage. While the packets are moving down the hierarchy, after inheritance at each frame, its slots may be compared with the slots of those frames obtained in the first stage. If the match is successful, we may then concatenate this frame with every frame brought to it by the packet. However, the result of each concatenation in each frame represents only a part of the resultant table and will still have to be returned to the controller.

The second option:

In this option, the user makes a query by giving only the common slot-names, which are then propagated through out all the hierarchies in parallel. While packets are traversing down hierarchies, after the inheritance operation at each frame, the common slot-names are matched with the slots of that frame. If the match is successful, the full version of that frame is returned to the controller.This operation continues until all the packets have reached the leaf-nodes. When the controller receives all the appropriate frames, it can then perform frame concatenation, similar to that set out above.

The following is a query made on the examples in figure 4.16 and 4.17:

```
( (hierarchies 'A' and 'B')
  (?frame-names
  (City-name (value ?V1))  ) )
```

After the propagation, from both hierarchies the full version of each frame that contains the common slots will be returned to the controller.

```
((Frame-name : Terry      ((Frame-name : Dary ((Frame-name : Ian
  (Emp-no : E107)           (Emp-no : E912)      (Emp-no : E239)
  (City-name : London))     (City-name : Leeds)) (City-name : Leeds))
((Frame-name : W34        ((Frame-name : W92
  (City-name : Leeds))      (City-name : Leeds))
```

Similar to option 1, the controller will perform the concatenation process and will return the result to the user.

Comparison and conclusions

In order to assess the suitability of join to the SM's computational model and architecture, the performance of the SM in relation to operations involved in join should be measured. As indicated above, there are two options that could be implemented in the SM.

The operations involved with the first options are as follows:

-Start packet propagation from the root-node in the first hierarchy. The contents of each packet is : ([common-slots] [upward-pointers] [inherited-slots])

-While the packets are moving down, at each frame perform the following:

check if the common slot-names are contained in that frame

if True, then perform inheritance and return the complete frame to the controller and then continue the propagation

if False, do not perform inheritance but continue the propagation

-Continue the propagation until all the packets arrive at the leaf-nodes

-When the first propagation ends, start the second propagation from the root-node of the second hierarchy. The contents of each packet are:

```
(  [((common-slot1  (value  v1)          ;note that each common
     (common-slot2  (value  v2)          ;slot has a value that has
     .                                   ;been obtained from the
     .                                   ;first  propagation
     (common-slotn  (value  vn)))]
   [upward-pointers]     [inherited-slots]  )
```

-While the packets are moving down, at each frame perform the following:

check if the common slot-names and their values are contained in that frame

if True, then perform inheritance and return the complete frame to the controller and then continue the propagation

if False, do not perform inheritance but continue the propagation

-Continue the propagation until all the packets arrive at the leaf-nodes

In the second option, the following operations are involved:

-Start packet propagation from each root-node

the contents of each packet are:

([common-slots] [upward-pointers] [inherited-slots])

-While the packets are moving down, at each frame perform the following:

check if the common slot-names are contained in that frame

if True, then perform inheritance and return the complete frame to the controller and then continue the propagation

if False, do not perform inheritance but continue the propagation

-Continue the propagation until all the packets arrive at the leaf-nodes

In both options the controller receives the full version of all the frames that contain the common-slots, after receipt of which it will apply the concatenation process (as explained above) to produce the appropriate products of all those frames that satisfy the given condition.

Note that at the implementation level most of the operations set out above are similar to those of query-object number 1 (see section 5.611, chapter 5). In this query, the frame-name is given and the system will return its complete version, which has undergone all the necessary operations.

Similarly, the operations involved with both options start with the packet propagation with each packet containing the common slot-names. While the propagation is in process, the inheritance and matching operations are performed on every mapped frame in a cyclic fashion, as described in chapter 4. Before the inheritance operation on each frame, its slots are matched with the common slot-names, and if the match is successful, after performing the inheritance, the full version of that frame will be sent to the controller. Whatever the result of the match, the propagation will still continue until the leaf-nodes are reached.

There are two major differences between the two options discussed above. Firstly, in option one, the propagation is performed twice, whereas in option number 2 the propagation is performed once only. Secondly, in option number 1, the number of frames returned to the controller is much smaller than those frames returned to the controller in option number 2.

Further, there is a major difference between operations involved with options 1 and 2 and those involved with query-object number 1. In both options, the propagation continues until it reaches the leaf-nodes, whereas in query-object number 1 the propagation stops as soon as the target-frame is found. However, as with options 1 and 2, the intensity of computation in query-object number 1 is at its highest level only on those paths that contain the query-related frames (see chapters 5 & 6 and, appendix D). This is due to the concept of locality, where for any query most of the related frames are clustered in the same area.

It is feasible, therefore, that the time penalties associated with query-object number 1, could also be utilised[2] for operations involved with option 3. In section 6.51 chapter 6, the test-runs for query-object number 1 are discussed.

In section 6.55, it was shown that, for each query, an average of 10 frames are processed. We may assume therefore, that for a join an average of 10 frames would be involved. If we further assume that the size of each frame is approximately 200 bytes (see chapter 6), we may then reach the conclusion that the time taken for 10 frames

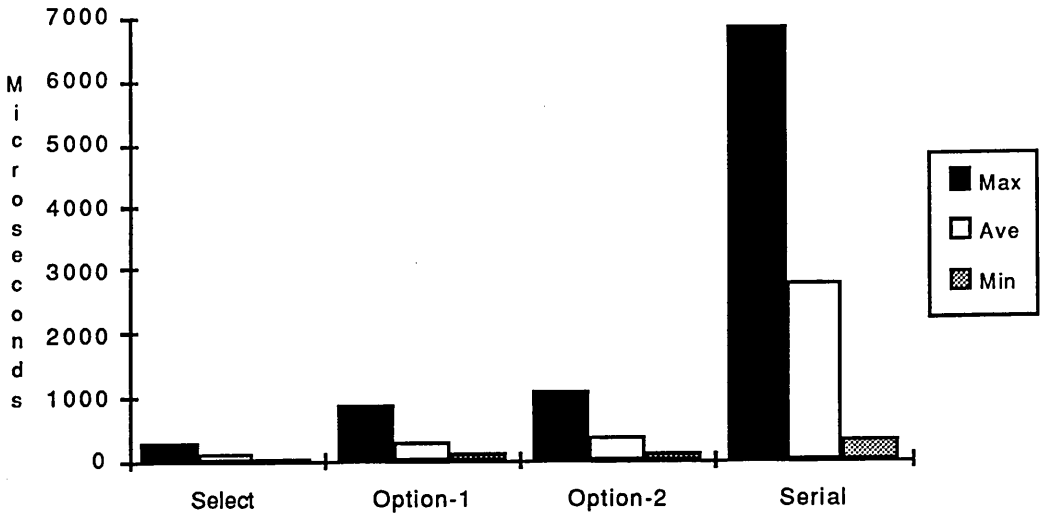
---

2 Here, only the time penalties for those queries that have asked for the leaf-nodes are considered.

to be transferred from the relevant PEs to the controller is approximately equal to 200 μsecs[3]

The time taken for concatenation may be calculated from the assumption made in section 6.2 chapter 6. The average time taken for concatenation and producing a product of 10 frames is approximately 400 μsecs (on the basis that it takes 100 nsecs for comparing one byte and 100 nsecs for loading).

These figures enable us now to provide a comparison between the time penalties incurred by operations involved in option 1, option 2, and their serial equivalent. In section 4.73, chapter 4, it was shown that the operations involved with select and project, will impose no extra time penalty on the SM's overall performance and are very similar to that of frame-related queries. Thus, by adopting the time penalties on frame-related queries for select, figure 2 shows the comparison between options 1 and 2, select and their equivalent serial operation.



	Option 1	Option 2	Select	Serial
Max	861.58	1123.16	261.58	6817.14
Ave	294.3	388.6	94.3	2766.06
Min	98.28	111.56	13.28	348.48

Figure 2.

From figure 2, we can demonstrate that the operations involved with select or its equivalent frame-related queries, are 27.5 times faster than their serial equivalent.

---

[3] Time taken for one frame to reach the controller = 200 X 100nsec/char = 20000nsecs/1000 = 20 μsecs (see section 6.2, chapter 6).

It can also be shown that the operations involved with option 1 and option 2 are 7.9 and 6.1 faster than their serial equivalent respectively. Note that the time penalties for both options one and two shown in figure 2, include the time taken for transferring all the appropriate frames to the controller, and for their concatenations.

Although the figures produced here are based on approximations, it is still possible to draw reliable conclusions. The conclusion that can be drawn is that join, in comparison to select, is a relatively expensive operation, due to its inherent serial characteristics (this is also an established fact with serial execution). However, option number 1, in comparison to its serial equivalent, still offers a gain in speed-up.

Further speed-up may be obtained by introducing a third option; an extended version of option number 1, which performs partial concatenation at the array level, and may offer more speed-up than any other option. The information given in figure 2, demonstrates that the select operator can readily be implemented in the SM, but the implementation of join is problematical in that time penalties will be increased.

Note that it is assumed that the result integration (concatenation of frames), which is the selection of appropriate frames from those returned to the controller on a specific condition, is performed at the controller level. This is the most efficient way of performing result integration, where the size of search space is sufficiently small to allow the controller to explore it in the most efficient way. By performing the result integration at the array level, on the other hand, the single propagation approach would have to be extended into a two stage approach. The first stage involves the propagation, the inheritance and matching operations; and the second stage will perform selection on a specific condition. It is certain that the number of frames to be explored in the second stage will be significantly smaller than in the first stage, which implies that a large number of PEs will be redundant while a small number of PEs are processing the relevant frames. One possible conclusion of this is that the amount of unnecessary communication and the inherent serial operations in the second stage will impose a substantial speed degradation.