MODELLING AND ANALYSIS OF PARALLEL INFORMATION SYSTEMS

Geoff Cutts

A thesis submitted in partial fulfilment of the requirements of Sheffield Hallam University for the degree of Doctor of Philosophy

July 1993

School of Computing and Management Sciences

Sheffield Hallam University

Abstract

This thesis presents an investigation of modelling and analysis of parallel information systems. The research was motivated by the recent developments in networks and powerful, low-cost, desk top multiprocessors.

An integrated approach for the construction of parallel information systems was developed which focussed on modelling, verification and simulation of such systems. The thesis demonstrates how Petri nets can be used for the modelling and analysis of entity life histories and parallel information systems, place transition nets for the modelling and analysis of entity life histories and coloured Petri nets for the modelling and analysis of complex parallel information systems.

These tools were integrated into a comprehensive framework which allowed for the modelling and analysis of complex parallel information systems and the framework was tested using a comprehensive case study.

The thesis concludes that Petri nets are an ideal tool for the modelling and analysis of complex parallel systems. Verification is possible with deadlocks and similar properties being easily identified. Further the transformation rules proved to be beneficial to the process of moving from one model to another. Finally simulation of parallel behaviour was possible because the underlying models captured the notion of parallelism.

CONTENTS

Chapter 1 BACKGROUND AND MOTIVATION FOR THE RESEARCH

1.1 Introduction

The development of a large computer application is, without doubt, one of the major consumers of resource in any organisation. Systems' development projects are invariably late, over budget, difficult to modify and maintain and do not meet the users' requirements. In addition as much as 70% of the total effort is spent on maintenance projects which leads to a dismal picture for systems' development as more complex systems embracing the notion of parallelism are developed.

Parallel programming skills were only required by a relatively small number of the software development community until the recent development of networks and powerful, low cost, desk top multiprocessors [Thakker-89]. It really does not matter whether an information system is to be implemented on a network or a parallel desk top computer, there remains a need to analyse, specify and develop parallel information systems. This need is addressed by this thesis. The hypothesis is that the approach to the development of parallel information systems is deficient in a number of key areas including modelling, analysis and verification, and simulation. This first chapter explains what is meant by parallel information systems, it describes the motivation for the research and concludes with a more detailed hypothesis.

1.2 Parallel Information Systems

Modern organisations exhibit parallel characterisitics, for example:-

1. An organisation is seen as one of interfacing individuals, all acting in parallel rather than as a unified whole.

2.There is non-hierarchical interaction as well as hierarchical interaction between groups.

3. Groups 'own' their information and software, and self control and

participation is to be encouraged.

4. Administration is decentralised.

[Burns-87].

The real world comprises people acting in parallel interfacing with other people both across as well as up and down management hierarchies. People use private as well as global data and they exhibit self control within a system of decentralised administration. Parallel information systems exist in many and various topologies, however, the physical structure of a network should not determine the communication pattern for the application. It should be possible to produce a logical design independent of the target hardware and software.

Parallel information systems fall into two categories; those which share a common memory and those where each node of the system has independent memory. These system categories have been defined as tightly or loosely coupled. This research is concerned with loosely coupled systems since they better represent the true world of information systems. In this type of system, programs execute on independent processors to achieve the goal of the information system. This requires communication between programs leading to information systems which encompass parallel processing. The motives for parallel information systems include performance improvement, cost reduction, configuration flexibility, increased reliability and ease of development.

1.3 Motivation

Many of the development problems stem from the failure to use a sound engineering approach to information system development. In this work the sound engineering approach is developed using modelling of the information system, analysing that model for consistency, prototyping, and simulation of the design before detailed building and testing.

Engineering is the profession in which a knowledge of the mathematical and natural sciences, gained by study, experience and practice, is applied with judgment to develop ways to utilise economically the materials and forces of nature for the benefit of mankind. The use of a sound engineering approach is what is lacking from current approaches particularly in the area of parallel information systems. Information engineers are responsible for constructing specifications, developing designs and implementing applications. Each of these may be regarded as a description of an application represented in some notation or notations. The development of notations, techniques and tools to specify and analyse parallel information systems is addressed by this thesis.

With the continual fall in the cost of microcomputers, and the steady advance in networking technology, it will become increasingly advantageous for commercial information systems to be implemented on a network of essentially independent processors co-operating to achieve the goals of the application. This work is therefore very important to the future work of information engineers. Current notations, techniques and tools fall considerably short when considering parallel information systems. If information engineers are to fulfil their role in this environment systems analysis and design methods and software engineering methods must be extended to parallel information systems in an elegant and efficient manner.

The more complex the application being designed, the more difficult it is to ensure that the design will work correctly under the specified conditions. The most complex design problem is that of distributed asynchronous parallel processes, which are at the heart of parallel information systems. Over the past few years, trends in the area have been characterised very much by developments in hardware, by slow progress in programming languages and software and by very little progress in information systems' analysis and design. This gap leaves a significant area of research.

Another issue is that of reliability. A centralised system based on a single controller is vulnerable to the failure of the control unit. Replication of this has previously been expensive, the minimal cost of components now makes it practical to partition the various functions onto separate processing elements. The complexity of individual functions is reduced, however, there is an attendant increase in the complexity of the inter process communication. This complexity exhibits itself in the problems of synchronisation and freedom from deadlock. If these cannot be organised properly then the designer will not be able to realise any of the advantages. This research addresses inter process communication, providing elegant analysis methods so that the benefits of synchronised but separate processing elements and reliability can be achieved.

1.4 Current Approaches to the Implementation of Parallel Information Systems

The most straight forward approach is to design and code a series of individual, separate programs with communications handled explicitly. This requires that the analyst, designer and programmer have knowledge of the physical communication mechanisms available. The designer must decompose the application into separate executable programs and specify the communication required between the modules. The disadvantage of this approach is that the designer can only use knowledge of the application, the target hardware and the target software as inputs to a design task based on experience alone.

My observation is that no acceptable approach is available to guide the decomposition task nor is there anyway other than code and test to analyse and review the results. A rigid structure is imposed on the application which can only be modified by a redesign exercise. Parallelism is considered, in this approach, after the analysis of the application has been completed. Parallelism is treated as part of design and implementation not part of the specification. This is the obvious disadvantage of the approach.

A single parallel program approach avoids the problem for the designer by considering the application to be one single distributable, potentially parallel program. The approach requires the designer to follow good design practice to ensure a highly cohesive, loosely coupled modular program. The designer may have an intuitive consideration for the parallelism inherent in the information system but generally the designer concentrates on 'good practice'. The programmer, therefore, is completely unaware of the physical communication media available it is the implementor who dictates the degree of parallelism at system set up. Again this approach does not capture the parallelism inherent in the information system.

It is the analyst and designer, not the implementor, who should control the degree of parallelism. The programmer should be unaware of the physical communication media available. The analyst should be able to capture the inherent parallelism in the application and this parallelism should be within the approach not 'bolted on' at some stage during the development process.

What is required, therefore, are specification and design schemata which embrace the notion of parallelism. The specification schema should specify what parallel opportunities exist within the information system. This requires the analyst to understand and document the degree of parallelism which exists within the current application and to represent the notion in the specification. The schema should specify the parallelism and communication between individual separate programs which comprise a single parallel information system. Further since programs are separated from each other the programmer may be unaware of the design schema leading to flexibility, reliability and ease of programming. This design schema should be produced as a product of a systems analysis and design exercise which from its outset considers parallelism. From understanding through to the production of a design schema the approach must embrace the notion of parallelism. The research is supported by developments in architecture which embrace parallelism. The development of a comprehensive set of techniques and tools to support the information system development cycle which embrace the notion of parallelism is highly desirable.

1.5 Developments in Architecture

Recent developments and innovations in hardware and software [Kung-89, Murakami-89] have resulted in performance improvement. Many developments have been characterised, among other factors, by the decreasing cost of components. This means it is now cost effective to realise parallel micro-computer based solutions to large complex commercial information systems. In addition to improved performance the motivation for their design includes simpler analysis, design and implementation, and greater reliability. However, although the information systems may show these improvements, new problems arise associated with inter-process communication [Munro-82]. This research sought to realise the benefits whilst providing a solution to the problems of inter-process communication.

An analysis of the architecture development cycle [Crookes-88] revealed a four stage cyclic process.

1. New hardware features are developed for improved machine(s) performance. This is usually followed by a period where programming is carried out in assembly language to exploit the new features and no change is made to systems analysis and design methods.

2. Machine oriented high level languages are developed to enable more convenient programming with still no change to systems analysis and design methods.

3. Machine independent notations are developed for the new concepts

provided by the hardware.

4. The notations are incorporated and supported by the hardware.

Only at this stage do language and methodology designers turn their attention to improved systems analysis and design approaches. Within the last ten years hardware designers have begun to support block structuring with stacks and register displays. Block structuring being originally developed by language designers. In addition it is only within that timescale that methodology designers have developed structured analysis and design development methods. A more recent example of language design leading hardware design has been the development of the Inmos transputer. The design of the transputer was influenced by the Communicating Sequential Processes principles. [Hoare-78]. New processor architectures are being developed at an ever increasing speed, there was never a more necessary time for technique and tool development than the present.

Many approaches are strong in their representation of structure, but weak when specifying processes. They usually produce a specification in terms of a composition of actions and data flows, but cannot reflect the intended behaviour in a dynamic process form. This imbalance needs to be redressed by augmenting representations and developing new representations to provide further process information [Kramer-88]. This gap is addressed by the research. A representation must be converted from one form to another during the development process. Ideally these conversions should be computer aided. In spite of the popularity of models and their support tools, relatively little work has been done in providing a theoretical approach to them. As a result transformation is done manually and is the source of many errors. Users tend to try to represent everything using one model. There is a need to develop a unifying theoretical approach [Tse-86]. A number of transformations are proposed in the thesis, each is given by a formal set of transformation rules which may be automated.

1.6 Conclusions

Traditional approaches to analysis and design are inadequate to move towards the implementation of parallel information systems. In particular methods of verification and validation are hopelessly inadequate. The result of relatively independent development of 'good practice', methods, techniques and formal notations has resulted in a tendency to prematurely freeze development. Information systems developed against the motives of increased performance, decreased costs, flexibility of configuration, reliability and ease of development lead to information systems which are parallel. The 'real' world of information systems is parallel, it is these properties which need to be understood and developed as an integral part of the information system.

1.7 The Hypothesis

1. Many information systems are parallel. There is a need to develop techniques and tools to deal with this parallelism.

2. Current structured systems analysis and design approaches are deficient in the modelling and description of dynamic states which are required to enable any approach to embrace the notion of parallelism.

3. There is a need to develop a means by which it is possible to prototype and implement information system designs for parallel systems.

4. There are a number of unresolved problems between the potential of parallel systems and their realisation in actual commercial feasible terms.

The hypothesis raises many problems which need solving, including:-

1. The gluing of techniques into an approach.

2. The modelling and analysis of dynamic states.

3. The development of software to enable prototyping and simulation of parallel information systems.

4. The development of tools to enable all levels of modelling with automated transfer from model to model.

The research therefore focused around the modelling, analysis and simulation of parallel information systems. There are a number of detailed problems whose solutions lead the way to techniques and tools to solve many of the problems.

Modelling was used to understand the breakdown and distribution of business processes and for understanding the structure of the interconnections. Verification was used to eliminate deadlocks and traps and for the identification of bottlenecks and simulation was used to examine the system's tolerance to breakdown as regards the ability to reconfigure and to assist the mapping of a logical design schema onto a physical configuration.

1.8 Objectives

The major objective was to develop a means of modelling, verification and simulation within an acceptable approach leading to the implementation of parallel information systems on parallel architectures.

The approach must develop a representation of the characteristics of parallel information systems. These characteristics may be grouped under four headings; structure, behaviour, communication and configuration.

Structure is the static partitioning of the information system into its components.

Behaviour is the dynamic operational characteristics of an information system or its components, representable as the flow of control between functions.

Communication is any interaction between system components during operation of the application. Communication has three aspects; static(interface), dynamic(protocol), and performance.

Configuration is the partitioning of the information system design into descriptions of building blocks for meaningful manufacture of system components.

The approach should be concurrent in two aspects:-

Development: Different organisational units should be able to develop information systems in parallel, large information systems being built from separate components developed as small projects.

Execution: Software components should be capable of being bound together at execution time with tasks performed in parallel on multiple processors.

Figure 1.1 summarises the objectives.

Objective	**Enabling technology**
Modelling of Entities and Events	Developed Data Flow Diagram
Entity Life Histories	Place Transition Nets
Information Systems	Coloured Petri Nets
Analysis of Entity Life Histories	The Theory of Place Transition Nets
Information Systems	The Theory of Coloured Petri Nets
Simulation of Information Systems	System Nets

Figure 1.1

In addition the research attempts to glue the modelling, analysis and simulation into a coherent approach using transformation rules wherever possible. For example rules were developed to transform entity life histories and entity event diagrams into coloured Petri nets and subsequently coloured Petri Nets into System Nets.

1.9 Thesis Organisation

Chapter 2 describes the research undertaken on related work in the area of approaches to information system development. Different methodological themes were researched together with their associated techniques and tools.

Chapter 3 examines two of the chosen techniques in detail, entity life histories and Petri nets. The approach introduces a number of novel features as well as using standard techniques such as data flow diagrams.

Chapter 4 describes the modelling of entities and events paying particular attention to the way in which they are developed from data flow diagrams.

Chapter 5 describes the modelling and analysis of entity life histories using place transition nets.

Chapter 6 looks in detail at the approach to the modelling of Information Systems using coloured Petri nets. It gives a set of formal transformation rules for the construction of coloured Petri nets from entity event models and entity life histories. Chapter 6 also examines the analysis and simulation of information systems using Coloured Petri nets and System nets.

The objectives included the development of a number of integrated tools to assist the approach. These tools are described on chapter 7. These chapters really describe the major contribution of this thesis.

A substantial system was developed to demonstrate the use of the approach and the support tools. This case work is described in chapter 8.

Finally chapter 9 concludes the thesis. It draws conclusions from the work undertaken as well as looking towards possible future work.

Chapter 2 RELATED WORK

2.1 Introduction

A wide ranging study was undertaken of existing techniques and tools which were considered to be candidates worthy of further study and development. This chapter documents that study, identifies candidates and selects and initially justifies those chosen for further study. A major decision was undertaken at this stage regarding the breadth and depth of research and study. It was agreed to limit the breadth to enable a number of promising areas to be studied in depth. The focus of the research enabled an early selection of promising areas for study.

2.2 Methodologies

Methodologies have been recognised over the last decade as one of the most popular approaches for information system development. They are widely accepted by practising computer professionals because of their largely top down nature and the graphical nature of the support tools. Methodologies have been designed by a number of authors and organisations using a wide variety of philosophies, frameworks, techniques and tools. Different approaches have been found to be appropriate for different environments and different stages in the development process. Methodologies may be classified under a number of themes.

Avison and Fitzgerald [Avison and Fitzgerald-88] considered a number of themes. The Systems Approach attempts to understand the nature of systems which are large and complex. This approach is mainly directed at gaining a better understanding of an organisation and a better understanding of the ill defined systems within an organisation. For the purpose of this research it is assumed that the objectives of the information system to be developed can be defined and therefore the systems approach may be considered to be outside the scope of the research. Whatever approach is adopted, the systems analyst ought to look at the organisation as a whole during the development cycle, this is the essence of the systems approach.

The Planning Approach stresses the planning involved within a development project. It sets out the phases of a project which might apply to any project. The objective of the research is to develop new techniques and tools which will fit within any overall approach. The planning approach is one approach that could be used by any organisation, however the exact approach used does not impact on the development of techniques and tools.

Avison and Fitzgeralds' third theme is the Participative Approach. This is an approach to development which encourages user involvement without specifying the techniques and tools to be used. Four qualities are suggested which are used later as design parameters for the new techniques and tools, they are:-

visibility - user understanding and control

simplicity - ease of use

consistency - a standard human computer interface

flexibility - the means whereby users can adapt the interface.

The Prototyping Approach is common in areas such as engineering where mass production makes it imperative that the design has been tested thoroughly first. It is also found in areas where the final version is one-off, like a bridge or building when it would be very expensive if the designers got it wrong.

The benefits to be gained from prototyping are obvious, increased quality and reliability with a reduction of risk and possibly costs. We should seek to realise these benefits in information systems development. Prototyping is therefore considered as a strong candidate for inclusion in my approach to modelling and analysis of parallel information systems.

Formal Methods are methodologies which incorporate mathematical precision in specification and design. The major advantage of formal methods is that results can be analysed and proved. Detailed analysis is a major goal of the research, formal methods must, therefore, be included.

Structured Methodologies are based on decomposition, that is breaking down a complex problem into manageable units in a disciplined way. To achieve a high degree of parallelism it is vital to break down an information system into its individual functions. Structured approaches have been studied to achieve this goal.

2.3 The Structured Approach

Structured approaches were introduced by many authors [De-Marco-78, Gain and Sarson-79, Yourdon and Constantine-78, Jackson-83] resulting in a number of structured systems analysis and design methodologies. These include Stradis, STRuctures Analysis Design and Implementation of Information Systems based on the work of Gain and Sarson; JSD, Jackson Structured Development; LSDM, Learmonth and Burchett Structured Development Method; SSADM, Structured Systems Analysis and Design Method, developed for the CCTA by Learmonth and Burchett and IE, Information Engineering based on the work of James Martin and Clive Finkelstein. Many of the benefits and objectives of the structured approach are embraced by the author's work on structured systems analysis and design, a basic input to the research.

Structured methodologies have many common techniques and tools. Methodologies themselves were therefore not studied in detail, the techniques and tools within the methods were the subjects for investigation. Nearly all structured methodologies embrace the notions of data flow diagrams and entity modelling whilst some include entity life histories and guidelines for selection of events and functions. A range of techniques within selected methodologies were therefore studied as candidates for further study and development. These techniques are discussed below.

2.3.1 Data flow diagrams

The data flow diagram provides the key means of achieving one of the most important requirements, that is the notion of the structure of functions which comprise an information system. The data flow diagram enables an information system to be partitioned into independent units for ease of general understanding and, more importantly from the viewpoint of parallelism, as an aid to understanding and specifying the inherent parallelism within the information system. Data flow diagrams also provide the analyst with the ability to specify functions at a logical level as well as providing the known benefits of enhanced user communication, abstracted levels of detail, etc.. One major benefit and reason for choosing data flow diagrams is their wide acceptance by practising systems analysts.

Data flow diagrams are however used in many different ways. There is considerable difficulty amongst analysts with two tasks, they are logicalisation and the identification of functions and events. It is my opinion that more formal rules should be developed for these tasks. This opinion has been formed from consultancy work and from teaching both within and external to the University. Discussions have taken place over the years with LBMS particularly on the above topics. Logicalisation of data flows diagrams was developed by the author [Cutts-87] and it is now used extensively by industry and academia. This research sought to extend data flow diagrams to a point where functions were atomic and data stores represent entities. This level of decomposition ensures the maximum potential for parallelism and leads to flexibility and ease of development.

Data flow diagrams were developed during the research to a position where data stores represent entities, functions are atomic and logical, and data flows represent entity access, messages between functions and basic information system input output. These diagrams were called entity event diagrams, they enabled the modelling of an information system, a prime objective.

2.3.2 Entity Modelling

Entity modelling is a particular technique used to analyse and understand the data within an organisation. The perceived requirement for the research was the development of entity life histories. Any technique leading to the identification of entities for inclusion on entity event diagrams and entity life histories will satisfy that need. Since there are many very acceptable approaches to data modelling of this nature further study and research was not considered necessary.

2.3.3 Entity Life Histories

The development of entity life histories requires that the effect of each event on the affected entities is charted. This is usually accomplished by some sort of cross reference chart or matrix. Methodologies such as SSADM require the development of two matrices, the function event matrix and the entity event matrix. The problems associated with this process include the identification of events which trigger processing and the subsequent matching of events against entities. The task often results in a major revisit of analysis which is not necessary.

The development of entity event diagrams as a natural development of data flow diagrams overcomes the above problems. It is my experience that the development of an entity function matrix is now very straight forward and it can be automated. The construction of an entity function matrix is a necessary first step towards the development of entity life histories. Entity life histories were first used by the author when working with a large British car manufacturer. They were used as a prime vehicle for developing and agreeing specifications with users.

Entity life histories gather together all the functions which effect an entity. Each row of the entity function matrix shows how an entity is affected by its functions. It does not show the sequence nor when it is valid to carry out a function. Entity life histories therefore link together entities and functions to produce a dynamic view, vital to the research, as opposed to the static views shown in data flow diagrams and entity models. Data flow diagrams tend to show the normal functions which affect an entity, entity life histories may be used to discover and document all error and abnormal processing.

Entity life histories became central to the approach and therefore much of the research centred around them. Their inclusion was very necessary because they provided the encapsulation of entity and functions required to achieve maximum parallelism and to provide a way forward to system modelling and implementation. Entity life histories were used to model the effect of events on entities, a dynamic view as opposed to the static views provided by entity event diagrams and entity models. Further with the development of place transition nets as the notation for entity life histories then analysis was possible realising two prime objectives, the modelling and analysis of entity life histories.

2.4 The Prototyping Approach

Prototyping became more common with the introduction of software tools such as screen painters, report generators, fourth generation systems and operating systems. Prototyping can be used as a technique to assist user analyst communication during many of the analysis tasks. For example the definition of screen output will be enhanced by the user being able to define the output using a screen and appropriate user friendly software. Secondly prototyping can be used during design to try out alternatives before the final design is defined.

Prototyping can, however, be more than just a technique. It can form the basis of a approach. It can be used to specify the required information system and to complete the design and implementation stages, by developing and implementing successive prototypes. Prototyping will be used within the overall approach to enable small scale systems to be built and tested with final system construction being done only at run time.

2.5 The Formal Methods Approach

Professionals who have become skilled in one or more of the above approaches appear to dislike formal approaches. There has been a slow acceptance of formal approaches themselves. However, it is plain that most formal approaches are merely notations. For example Z and VDM (set theory), OBJ (heterogeneous algebra), CCS and CSP (process algebra), Petri nets (graph theory) and Prolog (Horn clause logic) all have their foundations in mathematical notation. The engineering approach desired requires the use of mathematical notations. A number of formal notations were therefore studied.

The main criteria for selection of a formal notation were the ability to deal with parallelism and the availability of a non mathematical interface to enable information system designs to be more easily understood and therefore studied. Further the chosen method should have a wealth of theory leading to analysis methods already researched. Finally any chosen method must be in a development arena which will enable this research project to prosper from that research. Only Petri nets meet all of the criteria.

2.5.1 Petri Nets

C. A. Petri developed a system for graphically portraying asynchronous events known as Petri nets. Net theory is a theory of system organisation which originated from Petri's dissertation [Petri-62]. The dissertation explained the need for a theory of asynchronous machines. Many of the early papers in the area were by Petri. The MIT Information Systems Theory Project saw the involvement of MIT in net theory. Many examples of early net theory using Place Transition nets [Reisig-84], a special class of Petri nets, can be found in MIT papers [Holt-70, Hack-73, Commoner-71, Commoner-72].

The application and theory of Petri nets has been further developed by many centres especially throughout Europe to include Higher Order Petri nets and complex and detailed mathematical analysis techniques and tools. H. Genrich and K. Lautenbach introduced system modelling with high level Petri nets in 1981 [Genrich and Lautenbach-81]. Additional modelling power was provided by introducing individual tokens, predicates on transitions and variables on arc labels. This enabled additional information to be embedded in the net. These models were called Predicate Transition nets.

Mathematical analysis of Petri net models is one of the major advantages of using Petri nets. The variables in Predicate Transition nets yield difficulties during analysis, this led to K. Jensen's variable free nets with individual tokens introduced in 1981 [Jensen-81] called Coloured nets. A full theory has been developed for Coloured nets [Jensen-93].

Other net models followed including FIFO nets [Roucairol-86], Timed nets [Ajmone-Marsan-84] and Stochastic nets[Pagnoni-86]. FIFO nets possess token characters which behave according to first in first out principals. The FIFO principle extends the power of place transition nets to that of a Turing machine. Timed nets provide a way to estimate response times in systems that have strict timings. This can be a very complex process, simplification of complex systems have therefore been proposed which preserve information for accurate response time estimation [Chung-88].

Timed Petri nets can represent real time asynchronous and concurrent activities with specific execution times and its execution exploits the maximum concurrency at a time during the execution. Timed Petri nets have been used for this type of analysis.

[Ramamoorthy-80, Zubereck-80, Maggot-84, Halliday-85, Coolahan-85]. The underlying idea of stochastic Petri nets is to apply the methods of stochastic process theory to a system represented by a net instead of the system itself. There exists a number of classes of stochastic Petri net. [Florin-81, Dugan-84].

Petri nets were developed to impose a communications discipline on asynchronous operators based on local conditions, and therefore to eliminate the need for global evaluation of the system state. This makes Petri nets an ideal notation for the modelling and analysis of parallel information systems. Petri nets were studied as a candidate for formal documentation of system designs. Their graphic syntax makes them ideal for understanding and developing designs. Their mathematical foundation makes them ideal for formal modelling of designs leading to analysis and optimisation prior to construction. Their ability to elegantly model parallelism makes them ideal for modelling parallel environments.

Many designers of parallel information systems have found Petri Nets very difficult to use as a design method. Petri Nets are a notation for expressing designs which embrace the notion of concurrency. Significant progress has been made recently by the Meta Corporation[Meta-Software-88] with the developed of Hierarchical Coloured Petri Nets supported by very sophisticated software tools. However it is also easy to observe that many of the published papers detailing significant use of Petri Nets for designing parallel systems have included very significant input from Meta[Huber and Pinci-91]. The acceptance of Petri Nets as a specification and analysis notation has been very slow. There has been for several years a need to concentrate on the application of Petri Nets rather than further development of detailed and largely unusable theory. This thesis fills the gap. Petri nets will not be acceptable to the information engineer until transformation rules from well established approaches such as the structured approach to systems analysis and design are developed and implemented as easy to use tools. For this reason this thesis developed transformation rules from recognisable structured analysis notations to a Petri net notation. In my opinion this is a major contributor to the reluctance to use Petri nets and the techniques and rules developed in this thesis contribute a major step forward for information engineers. The study described above led to the identification of a number of key techniques which were considered as strong candidates for further research. In addition research was carried out into the area of proposed extensions to current techniques.

2.6 Extensions to Current Techniques

There are currently several approaches to the development of parallel information systems [Munro-82, Aspinall-77]. There are approaches where the problem is seen as a set of communicating processes and those based on data flow.

Extended data flow diagrams are one approach where events are shown by dotted data flows which originate either from an external source or from a special function within the data flow diagram. These data flows trigger or stop the execution of their destination function. This notion was used in the development of entity event diagrams.

State transition diagrams have also been used to show how a system changes its behaviour in response to outside events [Post-86]. These diagrams were used as an extension to the Yourdon Methodology to incorporate real time process control. State transition diagrams are a special case of Petri nets, Petri nets without the parallel construct. The ideas presented in this research have been taken forward to the development of the new approach.

The methodology also uses data flow diagrams for decomposing problems into inter-communication sub-units, mini-specifications for procedure specification, a data dictionary, and entity relationship diagrams. Events may arrive from outside the system by direct signals, thus the definition of an event is an atomic happening which changes the state of the system which can be further represented on a state transition diagram. These state transition diagrams are a simpler form of entity life histories, particularly entity life histories which have atomic events as their functions.

A design methodology, CYBA-L [Munro and Dagless-82] partitioned the problem into a control graph and a data graph. The control graph specified the sequence of operations in the data graph. The methodology used stepwise refinement to move from an abstract level of control to a final design specification incorporating concurrency. This method leads to two separate but similar diagrams.

Process Graphs [Jelly, Gorton, Grey-93] are a language and architecture independent graphical design notation for parallel software systems. Designs are constructed by expressing the partitioning of the problem into processes and then specifying the communication relationship between them. The approach requires the programmer to understand and specify how, when and why the processes interact. In my opinion leaving decisions of this magnitude to the technical does not achieve the objective of harnessing the parallelism inherent in the information system itself. It merely seeks to increase throughput on purely technical considerations. The authors state there is no single approach which has tried to encompass analysis, proof and verification in the systems development cycle for the building of parallel software. The authors' approach is to decompose the problem using the process graph notation then to translate the process graph to a Petri net. Additional elements have to be added to the net model to show the internal operation of processes. The Petri net is then used for validation of the design before translation into an implementation strategy [Gorton-93].

This work is similar to my research, it concentrates on program design whereas my concentration is on information system design. There is a real need to bring the research into the information systems arena and out of the technical.

2.7 Breaches of Current Methodologies

There are many breaches of current methodologies which should be avoided. Logical analysis is difficult and seldom done, the processors and their functions, for most information systems, are decided upon for reasons of implementation necessity before analysis commences. The architecture and allocation of functions to parallel processors is highly influenced by the configuration of hardware considered necessary for the information system and by previous similar information systems. This naturally influences the way in which data flow diagrams are constructed based upon architecture considerations rather than by doing an independent analysis. Once the architecture of individual parallel processes has been decided there is a tendency not to continue the use of structured techniques into sequential process design. The approach must guard against the tendency to short cut.

2.8 The Environment of Parallel Information Systems

To enable an objective selection of techniques for further study it is necessary to define the environment of parallel information systems. An early definition was given in chapter 1, the next few paragraphs seek to further refine that initial definition.

The environment of a parallel information system can be considered to be two fold. The external environment which may influence the information system and the environment of the information system itself. The information system environment covers a structural dimension, a geographical dimension, a temporal dimension, an activity dimension, a spatial dimension, an economic dimension and a social dimension. Each of these is defined below.

Structural dimension, the management and staff structure working within the information system environment. Very often a hierarchical structure dominates with information systems contained within the levels.

There is a need to link information systems across levels and up and down heirarchies.

Geographical dimension, the physical placement of resource including personnel across an organisation.

There is a need to allow information systems to operate across departments, buildings, counties, states, and countries.

Temporal dimension, the hours of work or hours at work.

There is a need to ensure people can work at their own speed, within their own time frame; major corporations have global operations which never sleep.

Activity dimension, the tasks which are performed.

There is a need to ensure each task making up the whole is defined and available across the network wherever and whenever it is required.

Spatial dimension, the area where a person works relative to their colleagues.

There is a need to enable people to work independently within their own work space or to work from any terminal world wide.

Economic dimension, the economic criteria which drives the organisation and by which people are assessed.

There is a need to develop and implement information systems in manageable affordable phases across the global network.

Social dimension, the social and psychological reasons which motivate people.

There is a need to allow people to interact with the system in a way which allows them best to be motivated by their working environment.

All these issues are addressed by the research.

People work in conjunction and co-operation with other people according to a structure and methods which have developed in time. People are, therefore, performing tasks individually or in groups using predominantly local resources. Tasks are performed in parallel using local resources including data to feed other tasks within the total, forming the organisational system. The organisations system is a set of concurrent tasks cooperating according to a structure to achieve an organisation's objectives. The processing and data is distributed across the organisation. Each of the dimensions described must be embraced by new and enhanced techniques within the new approach.

Parallel information systems are systems comprising software, hardware and people which exhibit parallelism, that is, the information system may be implemented on a multi-processor configuration using distributed data. A typical information system might be car hire where the software is implemented on personal computers linked within each car hire station by local area network and linked nationwide by a wide area network. Each local area network stores the data concerning hire vehicles currently available from that station with each personal computer across the network capable of implementing, in parallel, any of the tasks required by the car hire information system.

The development of parallel information systems requires analytical ability to be able to detect effects such as deadlock. Predictive ability is required to be able to predict response times for different system configurations and synthetic ability is required to be able to bring together different views of an information system to produce the composite design schema for the complete information system. The ability to perform all of these tasks must be within the new approach.

2.9 Requirements for the Approach

It is necessary to fit the new techniques and tools into an overall approach to development. Three main themes describe the overriding requirement for any new approach. They are a better end product, a better development process, and a standardised process. [Avison and Fitzgerald-88].

A Better End Product.

The quality of information system produced is very difficult to judge. The following points might be considered when examining quality. Key points are underlined.

<u>Acceptability</u> *<u>Availability</u>* *Cohesiveness* *Compatibility* *Development rate* *Documentation* *Ease of learning* *Economy* *Efficiency* *Feasibility* *Fitness-for-Purpose* *<u>Flexibility</u>* *Functionality* *<u>Implementabilty</u>* *<u>Low coupling</u>* *Maintainability* *Portability* *<u>Reliability</u>* *<u>Robustness</u>* *Security* *Simplicity* *Testability* *Timeliness* *Usability* *Visibility*

A Better Development Process.

The benefits which are achieved by a tight control of the development process and the identification of the deliverables at each stage are issues for this heading. The benefits include improved project control, improved productivity, decreased cost of development and a reduction in the skill level required for projects.

A Standardised Process.

This includes the benefits of having a standard common approach across an organisation. An approach was developed within the work to enable the techniques and tools to be applied to the case study.

A contractual model of product development was proposed by [Cohen-82]. He still considered the development process to be a sequence of stages; specification, design and implementation but with the boundaries between the stages being identified by deliverables. Each deliverable is looked upon as a contract. Figure 2.1 shows the stages and contracts. Further it shows the consequence of non agreement over the contract.

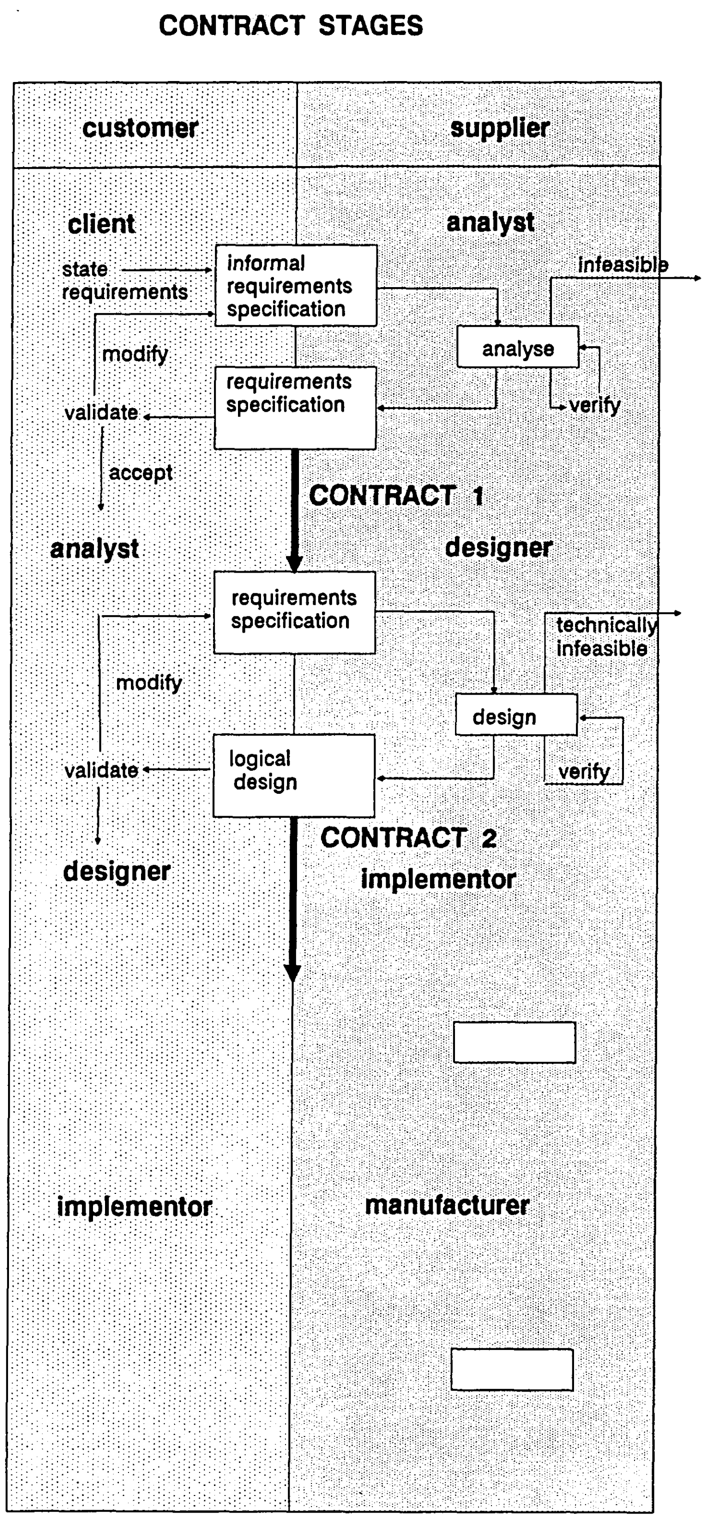


Figure 2.1

The proof of a contract is largely the acceptance of the contract by the parties involved. For example the first contract is between the client and the supplier for the product itself. The automation of correctness proofs could massively increase the quality of the development process. However to enable proof the specification must be sufficiently formal, the parties must be able to argue the correctness and the reviewers must be able to follow it. This is the approach followed in the development of the case study.

2.10 Detailed Requirements

This gives rise to a number of detailed requirements for the techniques and tools to be developed. These requirements are listed below.

1. Modelling serves as a means of communication between the users and the developer. By using various models and by continuous development of the models the current situation can be transformed systematically into a new situation. In this way it enables the developer to turn real world situations into manageable abstractions. Finally it forms a basis for implementation.

2. Functional decomposition is important to ensure a process oriented perspective which is understandable. We need to move away from any emphasis on a computerisable process towards an understanding of processes from the real world that could, if required, be computerised. Business activity is highly complex. To understand this complexity a top down approach is required with high level processes being represented by 'black boxes' which are decomposed into their more detailed processes. This leads to a hierarchical set of representations.

3. Encapsulation is the key idea of the object oriented approach in that an object should encapsulate its data and the operations on its data. The encapsulated objects in the new approach will be entities with their events charted by means of entity life histories. A

compromise is allowed in that the entity may be read but not modified by external events. In this way all entities are visible but protected from interference.

4. A graphical representation is a schematic convention for displaying the outcome of a technique. A graphical representation is required to aid understanding by everyone involved with the development process.

5. The term prototyping typically covers various possibilities from a rough and ready version of the information system to dummy screens containing illustrations of what and how the user will observe in the final version. Prototypes get the visible aspects of the information system correct. It is much more difficult to evaluate correctness, completeness and consistency of complex computation and integrity rule, therefore prototyping is used throughout the proposed approach.

6. Behavioral analysis focuses on the dynamic nature of the data and the need to understand events as they happen in the real world. It is necessary to distinguish between business activity and business events. Order processing is an activity and receipt of an order an event. Application areas are event intensive, careful analysis is required to determine the sequence or otherwise of events to protect the integrity of the data. Behavioral analysis provides a cross reference between the process and data perspectives.

7. Formal methods have excellent expressive power and are well suited for making inferences about specifications. Verification becomes possible, however, formal methods are generally much harder to apply if only a formal language is used. Formal underpinning needs to be provided in a mixed form where there exists more than one form to represent the specification. Graphical representations play a major role with automated transition from graphical form to formal notation and vice-verca. Analytic

techniques give rise to the detection of inconsistencies and incompleteness. Analysis of entity life histories leads to the detection of deadlocks and traps and the identification of process loops. Analysis of Coloured Petri nets leads to the detection of deadlocks, traps and bottlenecks in the final system.

8. Simulation is a means by which a prototype may be made dynamic. Simulation will be used as a means of testing the complete information system on a single processor prior to implementation on a multiprocessor system. Large complex systems can only be constructed and enhanced by synthesis of small system building blocks. The techniques used within the approach must ensure that when the blocks are cemented together the correctness of the whole can be guaranteed by the correctness of the individual parts.

The major thread of the proposed approach is to be able to transform a 'model of the real world' into a 'specification of requirements' into a 'realisation' of those requirements.

2.11 Conclusions

Systems analysis and design methodologies [De-Marco-78, Gain and Sarson-79, Yourdon-78, James Martin-80, Finkelstein-89, Jackson- 83, LBMS-87, LBMS-90, Downs-88, Cutts-87, Ashworth-90, Cutts-91] were developed to overcome many of the problems evident in the development of large complex computer information systems. Techniques selected from the structured systems analysis and design methodologies and engineering approaches coupled with concurrency theory seem ideal candidates to form the basis of a new development approach for modelling and analysis of parallel information systems.

It is my opinion that to support the development of parallel information systems new tools and new ways of implementing information systems are required to enable a new approach to information systems development. The research develops new techniques and new tools to enable the modelling, analysis and simulation of parallel information systems.

Chapter 3 The Approach to Modelling, Verification and Simulation

3.1 Introduction

The table below represents a summary of the conclusions reached in chapter 2.

Requirement	**Enabling Technology**
Modelling of Entities and Events	Extended Data Flow Diagrams
Entity Life Histories	Place Transition Nets
Parallel Information Systems	Coloured Petri Nets
Verification of Entity Life Histories	The theory of Place Transition nets
Parallel Information Systems	The theory of Coloured Petri nets
Simulation of	
Parallel Information Systems	The System net Environment

A further requirement was to glue the modelling, verification and analysis and the simulation into a life cycle type approach.

This chapter introduces modelling, verification and analysis, simulation and finally the overall life cycle approach. Before looking at each of the requirements a brief description of each enabling technology is provided. Readers who are familiar with these techniques may wish to omit this section.

3.2 Enabling Technologies

3.2.1 Data Flow Diagrams

Data flow diagrams are well known techniques. The research used the standard notations from SSADM. For these reasons it was not considered necessary to describe data flow diagrams in detail. The usage of data flow diagrams was extended to enable entities and atomic events to be discovered and documented. The proposed extensions are described in Chapter 4. They have been called entity event diagrams.

3.2.2 Petri Nets

3.2.2.1 Introduction

Petri nets can be used as a method for representing any given application to any degree of refinement. The simple and immediately understandable principles of application modelling with nets makes it possible to provide an illustrated description of the model without having to develop the mathematics behind it. This satisfies the need for a graphical interface to aid understanding of the specification.

Their graphic syntax is precisely defined[Reisig-84], they are strict "bipartite graphs" and semantics exist for them. Their mathematical foundation is straight forward, it is possible to fall back on formalisms to clear up doubts and sufficient theory is available, and in part implemented on computers, when it is needed. This satisfies the need for detailed analysis of the specification. Petri nets have been and are being used in many areas of data processing, for the modelling of hardware, communications protocols, parallel programs and distributed databases, particularly in the requirements specification context. Their graphic syntax coupled with the mathematical foundation makes Petri nets an ideal vehicle for specification and analysis of complex parallel information systems. Many approaches to the modelling of behaviour of an application result in a separate model, eg. CSP. In a Petri net approach behaviour of the application is an integral part of the design schema. This satisfies the desire to keep the number of notations to a minimum since it is necessary to specify transformation rules from notation to notation.

The concepts of Petri nets have been used for the modelling of database structures

[Voss-80, Voss-81, Voss-87, Hura Singh Nanda-86], specifically using the reachability and conflict concepts. Petri nets have been shown to provide useful data for the design of databases and insights into the behaviour of databases have been obtained by analysis of Petri net models. From the reachability set it is possible to study various problems such as safeness, liveness, boundedness, conservation, concurrency and performance. This shows the depth of research currently available within the Petri net community.

The problem of sequential program construction and manufacture has been widely studied but there are only a few studies on parallel program manufacture. Petri nets extended in several ways have been used to study parallel program construction [De-Bondelli-83, Cutts/Onley-88]. The modelling of Ada by Extended Petri nets was undertaken by [Ghezzi-88] resulting in the identification of unfeasible rendez-vous sequences. The use of Petri nets in the field of parallelism is not new, they have proved to be beneficial in many areas including the specification and validation of communication protocols [Diaz-83, Diaz- 87].

3.2.2.2 Properties of Petri Nets

It is worth noting the properties of Petri nets which are immediately relevant to the objectives.

1. An ability to describe information systems at various levels of detail [Meta Software-88].

2. A graphic and precise nature.

3. Theory and tools exist to determine and verify the dynamic behaviour of Petri net systems from their structure.

4. An ability to represent parallelism.

Petri nets attempt to give satisfactory answers to the following questions.

How can we separate different views and relate them to one another?

How can we model the (dynamic) behaviour of an information system alongside its (static) structure?

How do we represent individual and interrelated processes in parallel information systems?

The most obvious difference between Petri nets and other methods is the fact that they provide a representation of parallel dynamic behaviour. No mathematics is needed. This is because Petri nets were specially developed with an eye to intuitive understanding intended to promote and formulate intuitive knowledge. This is necessary since an essential part of systems analysis and design comprises communication between the user and the development team. The user cannot and should not be expected to deal with a complicated formal system. However, mathematics is indispensable where analysis procedures are used to establish the characteristics of a design. For these reasons two special classes of Petri nets were chosen for the research, place transition nets and coloured Petri nets.

3.2.2.3 Definition of Place Transition Nets

A formal definition of place transition nets is given below. This formal definition is required later for the definition of the transformation rules from place transition nets and entity event diagrams into coloured Petri nets.

A place transition net consists of

1. places, represented as circles.
2. transitions, represented as rectangles.
3. directed arcs from places to transitions.
4. directed arcs from transitions to places.
5. a capacity indication for every place, represented as a label.
6. a weight for every arc, represented as a number.
7. an initial marking, defining the initial number of tokens for every place (cannot be greater than the indicated capacity).

The arc weight "1" can be omitted or assumed. The capacity of a place does not need to be indicated if it is not important or if there is never a danger that it will be exceeded. Moreover a place can have an unrestricted (infinite) capacity. A marking of the net, such as the initial marking, is indicated by the number of tokens in every place. Further a place p is in the preset (or post-set) of a transition t if there is an arc from p to t (or an arc from t to p);

A transition t is said to be enabled if

> 1. for every place p from the preset of t the weight of the arc from p to t is not greater than the number of tokens indicated at p.
>
> 2. for every place p in the post-set of t the number of tokens at p increased by the weight of the arc from t to p is not greater than the capacity of p.

An enabled transition t occurs in that the number of tokens at every place p is decreased by g if $p \rightarrow t$ and in that the number of tokens at every place p' is increased by g' if $t \rightarrow p'$.

3.2.2.4 Reachability Trees

Reachability trees represent all reachable markings and therefore all routes through a net. The ability to analyse a net for reachability provides an excellent analysis technique for both entity life histories and system nets. Reachability trees allow the definition of dynamic properties which are related to occurrence sequences. An occurrence sequence is a sequence of valid functions which may affect an entity. In the research it is required that the first occurrence is the insertion of an entity and the final occurrence its deletion or archive. Further it is required that between the first and final occurrences the entity is 'live' and after the final occurrence the entity is 'dead'. Liveness and dead markings are two of the properties which can be analysed automatically using reachability trees which lead to the discovery of important conclusions regarding entity life histories.

A marking is dead ie. no function can execute, if no transition is enabled from the marking, and a net is said to be deadlock free if no reachable marking is 'dead'. Liveness is another required property. Liveness requires all places to be reachable from the initial marking and the net to be deadlock free. Although live and dead are related concepts, it should be noted that they are not negations of each other. Demanding a net to be live is much stronger than demanding it to have no dead markings.

3.2.2.5 Calculation & Interpretation of Place & Transition Invariants

Place and transition invariants will be used to further analyse entity life histories. They provide static structural analysis which is highly desirable when dealing with large nets.

There are sets of places on which the token count can vary in a particular way leading to deadlocks and traps [Hack-73]. Deadlocks are sets of places which remain empty if they loose all their tokens and traps are sets of places which remain marked once they gain at least one token. It is necessary for the liveness of a marking that all deadlocks remain marked. The motivation for analysis using place invariants is that we do not wish to analyse nets by reachability tree since in a large net this soon becomes unwieldy. This will be particularly important when we build large system nets from the synthesis of nets representing entity life histories.

Definition

Let $N = (S,T;F,K,W)$ be a pure place transition net.

1. A column vector $v:S \rightarrow Z$ indexed by S is called an s-vector on N.

2. A matrix $N:S \times T \rightarrow Z$ where $N(s_i, t_j) := W(t_j, s_i) - W(s_i, t_j)$ is called the incidence matrix of N. (The same letter is usually used for the incidence matrix and the Petri net.)

3. I is called a place invariant of N if $I^T . N = 0^T$

Theorem

Let M be a marking of N; then for all M' E [M] : $I^T.M = I^T.M'$. For a proof of this theorem see [Lautenbach-87].

Calculation of place invariants

Calculation of place invariants involves the solution of a set of linear equations generated by making $I^T.N = 0^T$. This set of equations can be solved using Gaussian elimination, by using defined algorithms [Martinez-82] or by using Petri net analysis packages. A review of packages available can be found in [Jensen-86].

Definition

Let N = (S,T;F,K,W) be a pure place transition net.

1. A column vector $w:T \rightarrow Z$ indexed by T is called a T-vector of N.

2. If J is a T-vector of N; J is called a transition invariant iff N.J = 0.

The transition invariant J represents the occurrence of transitions. For example $J^T = (0, 1, 2, 1, 0)$ represents

0 occurrences of transition A

1 occurrence of transition B

2 occurrences of transition C

1 occurrence of transition D

0 occurrences of transition E.

Theorem

If J is a transition invariant then there exists a marking M_j of N that is reproducible under J. For a proof of this theorem see [Lautenbach-87].

Place transition nets provide all the necessary and sufficient formalisms to adequately represent entity life histories. The use of nets for entity life histories is described in detail in chapter 5.

3.2.2.6 Coloured Petri Nets

Introduction

Coloured Petri nets, CP-nets, are a more powerful net type used to describe complex systems in a manageable form. Jensen has shown how Coloured Petri nets can be constructed using hierarchies [Jensen-89] and their applicability to real world systems [Jensen-87]. Shapiro proposed an extension to Coloured Petri nets that incorporates the concept of time [Shapiro-90].

The Coloured Petri net (CP-net) shown in figure 3.1 describes the system shown by the Place Transition net, figure 3.2. [Jensen- 91].

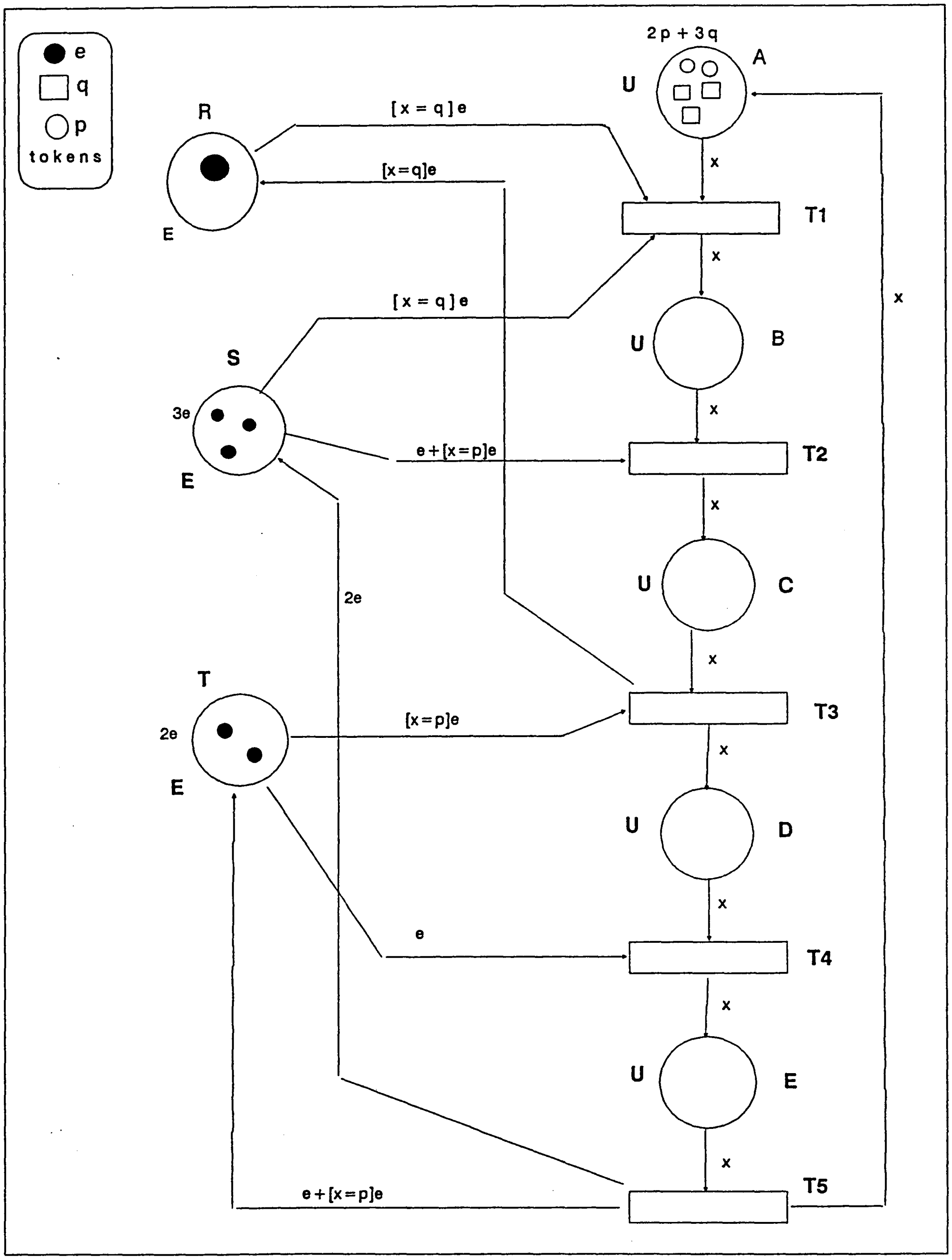

e
q
p
tokens
2p + 3q
A
U
R
[x = q]e
[x=q]e
E
T1
x
[x = q]e
U
B
S
3e
E
e+[x=p]e
T2
U
C
2e
T
2e
E
[x=p]e
T3
U
D
e
T4
U
E
T5
e+[x=p]e

Figure 3.1

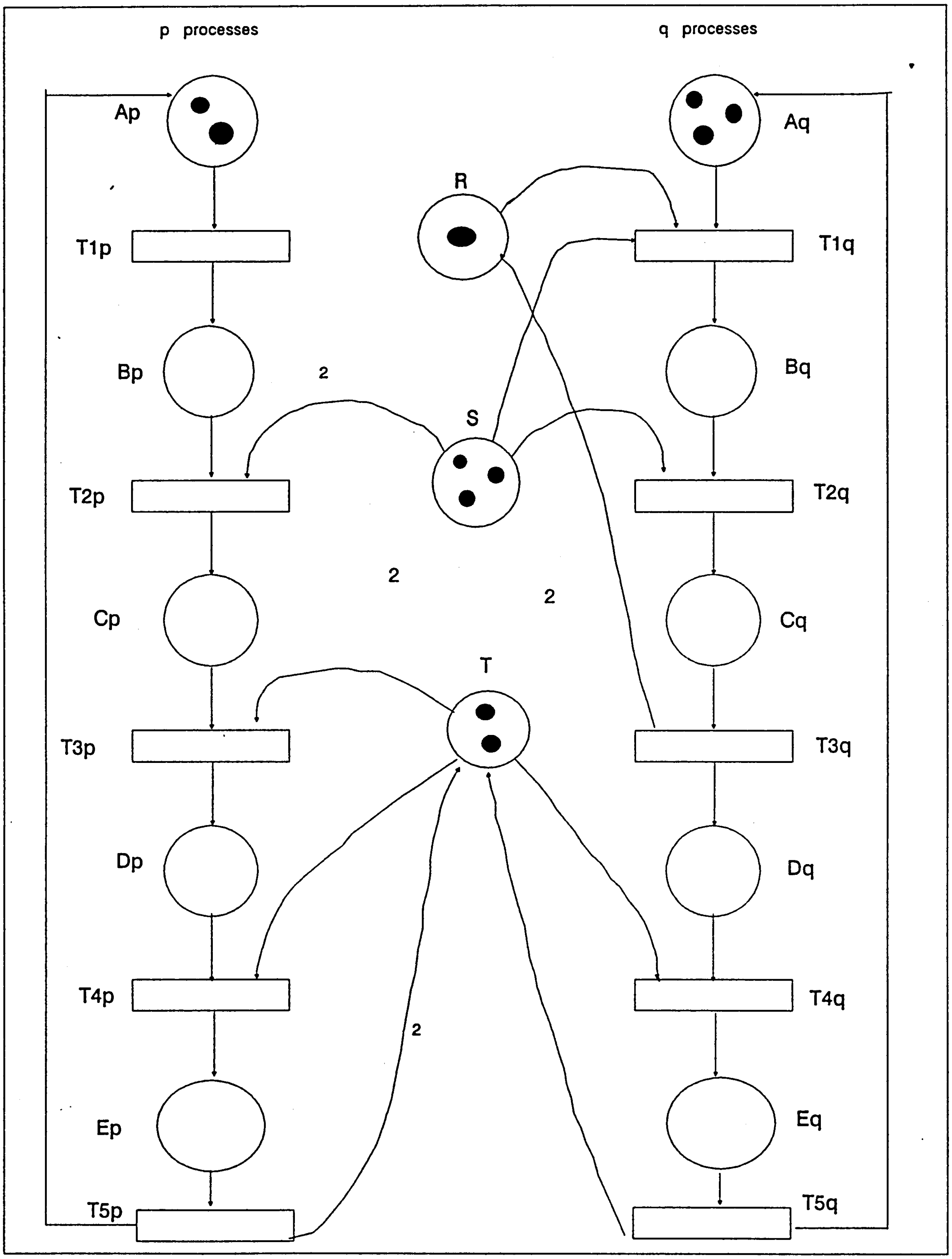

p processes
q processes
Ap
Aq
R
T1p
T1q
Bp
Bq
2
S
T2p
T2q
2
2
Cp
Cq
T
T3p
T3q
Dp
Dq
T4p
T4q
2
Ep
Eq
T5p
T5q

Figure 3.2

The places Ap and Aq are folded into one place. To distinguish between p processes and q processes different tokens are used in the coloured net. Squares represent q processes and circles p processes. Colours are often used to distinguish tokens. Each place has U = { p,q } as the set of possible coloured tokens. Place A has 2 p tokens and 3 q tokens, the marking of place A is the bag { 2p + 3q }. The places R, S and T only require one kind of token, therefore E = { e } is the set of possible token colours. The transitions T1 to T5 are also folded. T1 represents both T1p and T1q. Arcs acquire arc inscriptions composed of variables and expressions. In this case the variable x takes the value p or q. This corresponds to transition T1 occurring in two different ways representing a p process or a q process. The colour sets for the coloured net are U = { p, q } and E = { e }. The variable x : U.

Both Petri nets describe two processes competing for resources R, S, and T. There is one resource of type R, three of type S, and two of type T. There are five processes, 2 p processes and 3 q processes. The p processes each require zero of resource R, two of S at one time and two of R one after the other. All resources are returned at the conclusion of the processing cycle. The q processes each require one R resource and one S resource at the same time, followed by a further S and a T. The R resource is returned part way through the processing cycle and the S and T resources are returned at the end of the cycle. The p and q processes are represented by individual loops in the place transition net. In the coloured net p processes are represented by hollow circles, q processes by hollow squares and resources by filled in circles. The tokens become individual.

The behaviour of the CP-net is exactly the same as the corresponding PT-net. When T1 occurs with occurrence colour p, a p token is removed from place A and added to place B. When T1 occurs with occurrence colour q, a q token is removed from place A and added to place B, and an e token is removed from place R and added to place S.

To describe this CP-net we use the notation

$$[\text{ Boolean }]\text{ Exp } = \begin{cases} \text{Exp} & \text{if Boolean} \\ 0 & \text{if not Boolean} \end{cases}$$

We can now formally describe the arc expressions surrounding transition T1 as shown in figure 3.3. The variable x is of type U, it may therefore take the value p or q.

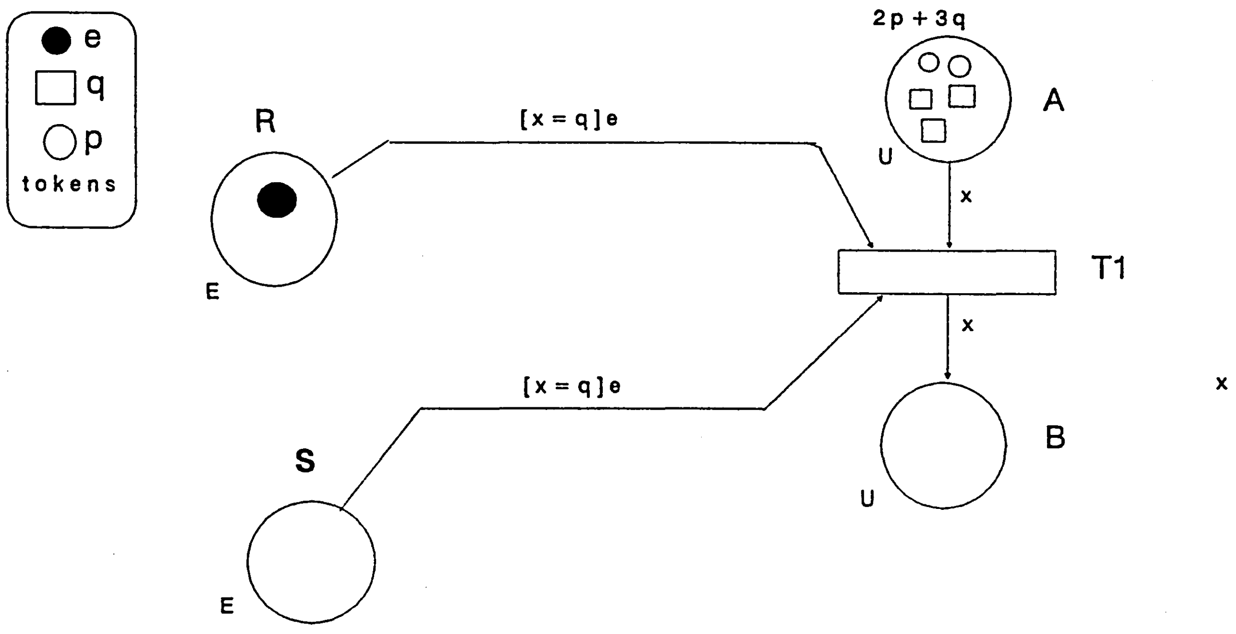


Figure 3.3

When transition T1 occurs with x = q then x = q is true and tokens of type e are consumed from places R and S and a token of type q is deposited on place B. The new marking is

place A 2p + 2q

place B q

place R 0

place S 2e

When transition T1 occurs with x = p then x = q is not true and no tokens are removed from place R or place S. A token of type p is deposited on place B. The new marking is

place A p + 3q

place B p

place R e

place S 3e

It is possible, as in this case, for a transition to be concurrently enabled with itself.

The Benefits of using CP-nets

The benefits of using Coloured Petri nets over place transition nets for large systems include:-

1. Description and analysis becomes more compact and manageable since the complexity is distributed between the net structure and the inscriptions.

2. Similarities between system parts are highlighted as having the same structure and inscriptions but different bindings.

3. Some syntax checking can be performed on the net such as consistency of arc expressions and colour sets.

The considerable benefits of coloured nets over place transition nets leads to their use for modelling and analysis of parallel information systems.

3.2.3 System Nets

System nets are a very simple realisation of systems which are specified using Coloured Petri nets. In System nets places represent relations and message buffers. They are drawn as shaded boxes and are implemented as relational tables using a relational database.

Transitions represent executable processes. They are drawn as rectangles and are implemented as two programs, a test program which determines if the transition is enabled and a further program which carries out the transition's task. The System nets are input to a simulation environment which provides a sophisticated work bench for developing, testing and simulating parallel information systems. Chapter 6 describes System nets and Chapter 7 the simulation environment.

3.3 Modelling

3.3.1 Introduction

The growing demand for information systems of ever increasing size, scope and complexity has caused the introduction of various high-level modelling languages, by which functional application requirements and information system components may be modelled at a conceptual level [Loucopoulos-92]. This research has developed the notion of entity event diagrams, Place Transition nets have been used for modelling entity life histories and Coloured Petri nets have been used for modelling information systems.

3.3.2 Entity Event Diagrams

Entity event diagrams are developed from 'conventional' data flow diagrams and entity models, both well known techniques within the world of structured analysis. Entity event diagrams are similar to data flow diagrams but restricted by asserting that each data store must represent an entity and each process must represent an atomic event. Further all processes and data flows must be 'logical', representing 'what' is required, not 'how' it is to be implemented. The diagrams are developed to represent the required system. Chapter 4 explains in detail the development of entity event diagrams.

3.3.3 Entity Life Histories

There are many notations for entity life histories, enhanced Jackson structures used in

SSADM, the network structures peculiar to LSDM and the structures based on Petri nets presented in [Cutts-87] and now used by many organisations. This chapter examines each of the structures and selects one for the further development of entity life histories.

SSADM uses a hierarchical notation for entity life histories very similar to the notation used for Jackson Structured Programming. Boxes represent functions with sequence being represented by placing the boxes left to right on the page. Repeating functions are annotated with an *, optional functions with an o, in the top right corner of the box. These constructs are easy to use for normal lives, however, to represent error situations and abnormal processing it is necessary to use quit and resume markers. These are difficult to insert, difficult to follow and there is no way to check that they are consistent. Finally there is no way to represent functions which may occur in parallel, except for a simple documentation 'trick' whereby the single line which represents sequence is doubled up to represent parallelism.

The notation was developed from a programming notation, it is therefore not appropriate for higher level design although many skilled analysts have used it for this purpose. Finally there is no mathematical underpinning to the notation which means that all analysis must be done by inspection. This is not sufficient in the arena of parallelism.

A network diagram convention was developed by LBMS. This convention overcame some of the drawbacks of the purely hierarchical notation. Different icons are used for insert functions, modify functions and delete functions making the diagrams clearer and a simple connection arrow shows the life flow. However the * is still retained for a repeating function and artificial perimeter boxes have to used to signify repeating section of the life history. Again, there is no convenient natural way to express parallelism, indeed parallelism is represented in the same way as alternate. No mathematical underpinning exists. It is interesting to note that the tools developed for structured systems analysis and design only provide a diagrammer for entity life histories, they are not loaded into the data repository for further analysis. This is partly due to there being no easy way to represent the conventions internally within the data repository. An important feature of entity life histories is the entity status. With the hierarchical and network notations the status is added after the completion of the diagram. This makes status identification difficult and often meaningless. All of the above difficulties are overcome using Place Transition nets, a subset of Petri nets, to represent entity life histories. Chapter 5 examines their use in detail.

3.3.4 Information Systems

The choice of Coloured Petri nets as the modelling technique for parallel information systems enabled the benefits of modelling with nets to be obtained without the side effect of large unwieldy models. The benefits of modelling with Petri nets have been discussed. Coloured Petri nets allow the modelling of complex systems in a manageable form folding places and transitions by making tokens individual and by including inscriptions on the net. The benefits were described in 3.2.2.6.

3.4 Verification and Analysis

3.4.1 Entity Life Histories

Formal verification of entity life histories is made possible because of the use of place transition nets which possess a formal mathematical notation. Verification tools were developed as part of the research and sophisticated tools are now available, for example from Meta. Verification was undertaken using reachability and linear algebra. Reachability allows questions such as 'Does the entity behave properly?', 'Is there always a route to end of life?', and 'Have all abnormal and error events been discovered?' to be answered. Analysis by reachability involves the execution of the net. This no longer becomes feasible with large nets, the so-called 'state space explosion' takes place. Static verification using linear algebra leading to the discovery of place and transition invariants allows deadlocks, traps and pathways to be investigated. Verification of entity life histories is described in Chapter 5.

3.4.2 Information Systems

Formal verification of information systems is possible because of the detailed theory developed for Coloured Petri nets. This theory developed over the past ten years has recently been incorporated into a sophisticated tool by Meta. Reachability analysis and place invariants lead to the identification of deadlocks and traps whilst transition invariants lead to the identification of pathways through the system. These pathways represent the transaction routes through the system. Further the recent development of timed Petri nets and their application to Coloured Petri nets leads to the identification of transaction bottlenecks.

3.5 Simulation of Information Systems

The transformation of information system models from Coloured Petri nets to System nets facilitates their implementation. The development of the simulation environment allowed the system to be incrementally developed and tested on a single processor. The simulation enabled incremental development and testing effectively implementing 'correct by construction'. Further the simulator enabled full system execution on a single processor whilst highlighting the full potential for parallel execution.

3.6 The Approach

The area of information systems is dominated by references to the real world and it has been argued that the problems found are a mixture of empirical, formal and engineering problems [Verrijn-Stuart-87]. Empirical problems are those concerned with observing real world phenomena, formal problems are concerned with abstraction, structure and representation of this knowledge in a way that it is possible to reason about it and engineering problems arise when one attempts to implement the information system.

[Loucopoulos-92].

It was argued earlier that the real world of information systems is parallel. The approach to development of parallel information systems is based on modelling for abstraction, structure and representation. The choice and development of modelling techniques was crucial to enable verification and analysis to take place and thus reasoning about the models. Finally a simulation environment was developed in order to minimise the engineering problems which arise during implementation.

Chapter 4 DEVELOPMENT OF ENTITY EVENT DIAGRAMS

4.1 Introduction

The objective of chapters 4, 5 and 6 is to explain the new techniques introduced as part of the research. Chapter 4 examines the development of entity event diagrams. Chapter 5 examines in detail the use of place transition nets for the development and analysis of entity life histories and chapter 6 looks at the transformation of entity event diagrams and entity life histories into Coloured Petri nets and System nets. The development of new techniques required for the new approach followed a series of guidelines developed to ensure good practice and commonality of approach.

Wherever possible graphical notations were chosen. The approach seeks to introduce increased rigour and formalism gradually from loose graphical representations during the 'understanding problems' stage through to exact mathematical representations in the design stages. Many methodologies suffer from problems in the transformation of information from one format to another. Transformation of this information is considered vital so, whenever possible, formal transformation rules have been developed and specified, and, in many instances, automated.

4.2 Entity Event Diagrams

4.2.1 Definition

An event is defined to be an atomic task which affects one entity only at a time. An atomic task undertakes a single indivisible action. Entity event diagrams are defined by their diagram conventions. Entity event diagrams comprise four icons, a rectangle representing an event, an open ended rectangle representing an entity, an ellipse representing an external source or destination for data and an arrow representing data flow. Entity event diagrams are a specific class of data flow diagrams.

The aim of entity event diagrams is to collect together all of the events which affect an entity. Since events are atomic there should be no possibility of further decomposition which will lead to maximum concurrency in the new system. If events are not atomic then opportunities for concurrency may be lost. Entity event diagrams are a development of data flow diagrams. They enable user- computer professional communication and ensure that the parallelism aspects of the business area are understood.

4.2.2 Construction of Entity Event Diagrams

A set of entity event diagrams may be constructed from a set of data flow diagrams and an entity model. An entity event diagram is a logical diagram developed from physical data flow diagrams. The transformation rules given below convert a physical data flow diagram to a (logical) entity event diagram.

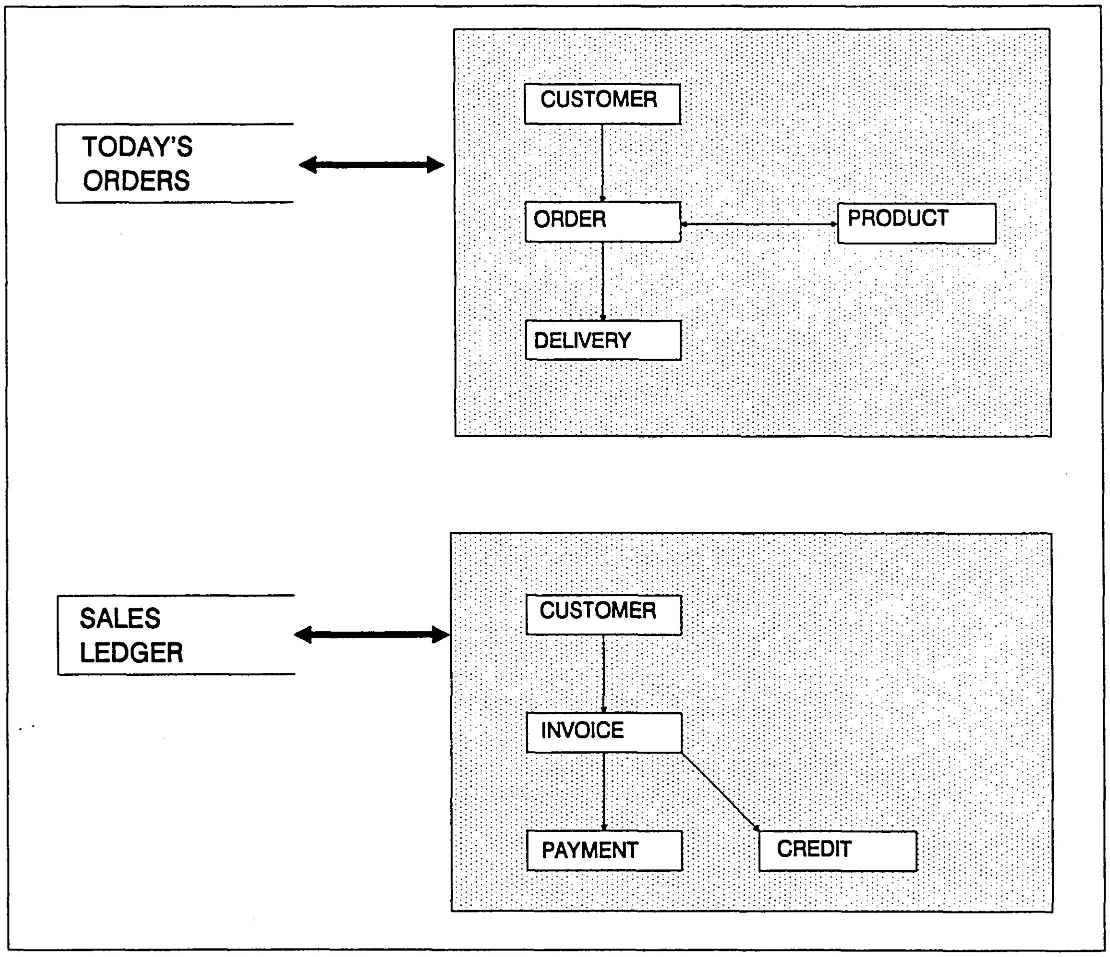


Figure 4.1

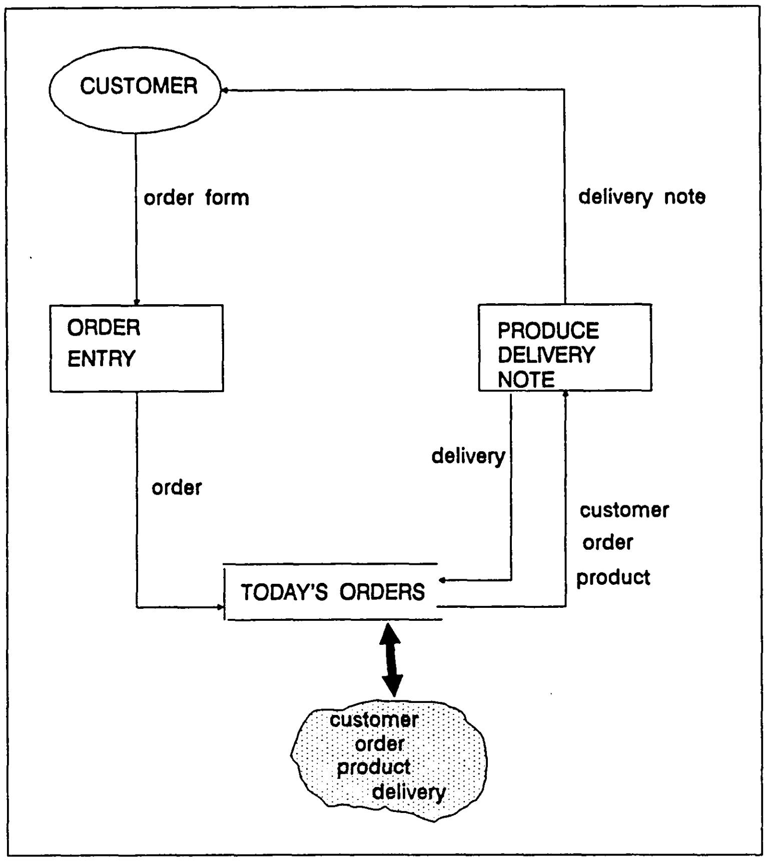


Figure 4.2

Transformation Rules

1. For each physical data store on the data flow diagram annotate the data store with the names of the entities it contains. An example is shown in figure 4.1.

2. For each data flow to or from a data store annotate the flow with the appropriate entity name or names shown in figure 4.2.

3. Delete the physical data store and terminate each data flow with an open ended rectangle named with the entity name.

4. For each data flow not to or from a data store, logicalise the name. For example:

reorder card might be renamed reorder request and sales printout, sales information. Other data flows might not be so simple, it may be necessary to undertake further investigation to ascertain the exact contents of the data flow. An example from a recent system was the change from 'reverse of white sheet' to 'testing instructions'. If the resultant name is that of an entity which flows between two events they must be decoupled. The entity must be inserted into the data store at the earliest opportunity with the subsequent events retrieving the entity for processing. This is to ensure maximum opportunity for concurrency.

5. Finally the DFD functions themselves must be logicalised and decomposed to create atomic events. Functions which perform physical only processing should be deleted and function descriptions should be modified. For example 'Print sales tab.' should be changed to 'Produce sales information'. This removes any reference as to 'how' the existing system operated. The function has now been transformed into an event, however, if the original DFD function had not been decomposed to a single action function, the resultant transformation may lead to multiple events. The transformation of functions to events must identify the atomic events within the function and it is these atomic events which must be recorded on the entity event diagram.

6. Some data flows may remain between events. These data flows have no data content, they are triggers indicating that processing has been completed by one event and another event may proceed. In this case the latter event cannot proceed until it is triggered. The data flow guards the latter event and is, therefore, termed a guard. An example of this type of transformation is shown in figure 4.3.

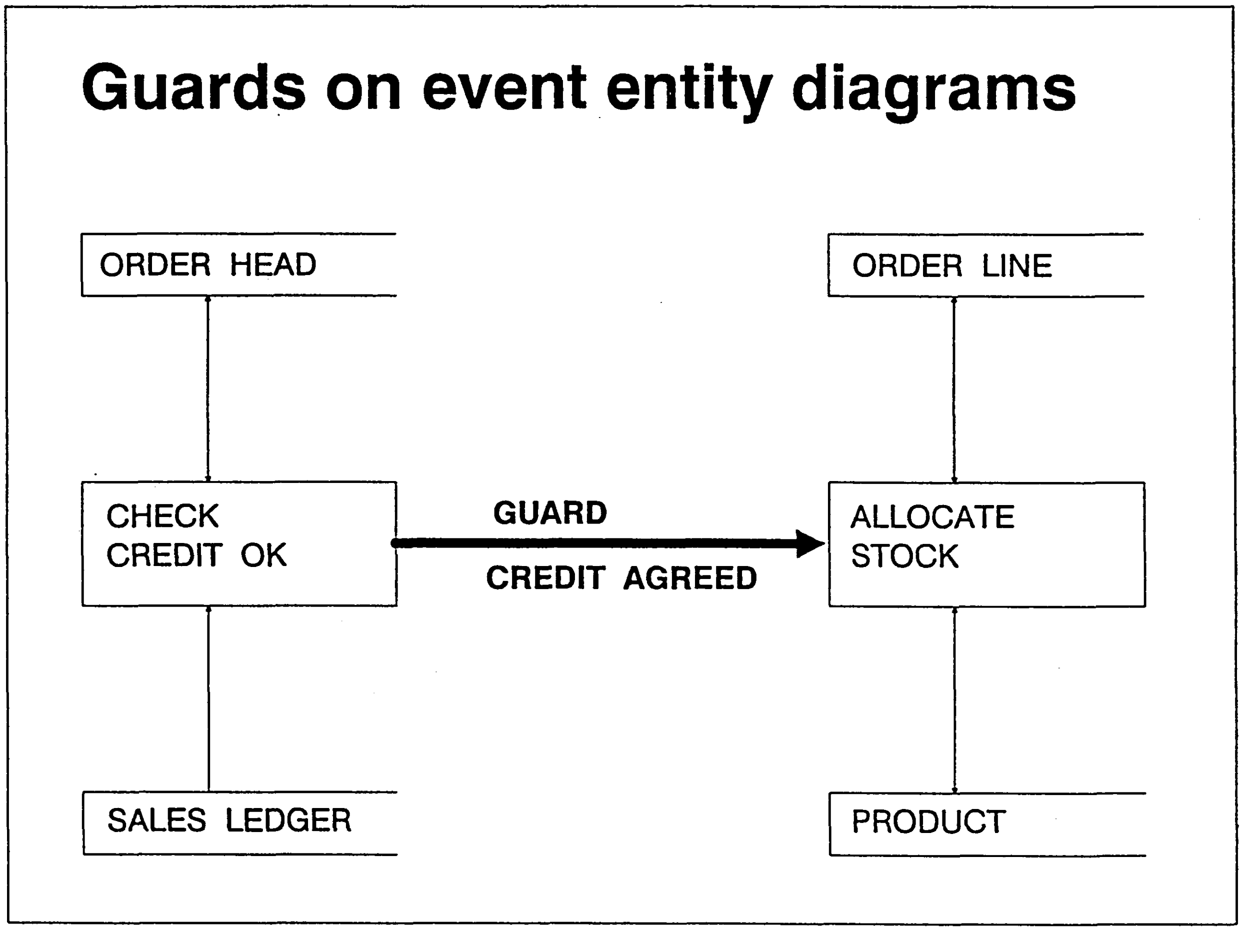


Figure 4.3

Entity event diagrams comprise

entities represented by open ended rectangles

atomic events represented by rectangles

data flows representing inserts, reads, modifies, deletes and archives of entities

data flows representing the information system's inputs and outputs

data flows representing guards

ellipses representing sources and destinations for data.

4.3 Conclusions

The transformation rules for conversion of data flow diagrams and entity models into entity event diagrams were developed using experienced gained from both consultancy and teaching activity and then further during the research. These rules are designed to maximise the opportunities for parallelism.

It was found necessary to develop rules for 'logicalisation' of data flow diagrams to enable practising analysts and students to convert physical data flow diagrams to logical ones. This fundamental task within structured approaches often merits one line in texts or courses, "Convert the physical dfd to a logical dfd". These rules have proved to be highly successful and are used world wide. The development of the guard comes from the observation that some data flows had 'no data'. The related research work also highlighted the need for guards which were generally documented using additional icons or by using separate diagram types. The transformation rules presented here have proved to be a natural extension of data flow diagram decomposition, they lead to inclusion of the entity model on the diagram, and they incorporate rules for 'logicalisation' providing an elegant development approach for entity event diagrams which is acceptable to the professional analyst.

This provides a major contribution to the research and the new approach.

Chapter 5 DEVELOPMENT AND ANALYSIS OF ENTITY LIFE HISTORIES

5.1 Introduction

The adoption of place transition nets for entity life histories was justified earlier. This chapter seeks to further justify their use by showing how to develop them and by demonstrating the many interesting properties which can be discovered using standard analysis techniques for place transition nets. The development of entity life histories has one prerequisite task, the production of the entity event matrix.

5.2 Entity Event Matrix

The construction of an entity event matrix is a technique which leads to the construction of entity life histories. The matrix has entities as rows and events as columns. The entities and events may be easily identified from the entity event diagrams. Figure 5.1 shows an entity event matrix.

EVENT → ENTITY ↓	ORDER ENTRY	PRODUCE ORDER ACK.	RECORD DELIVERY	PRODUCE INVOICE
CUSTOMER	R	R		R
PRODUCT		R	M	R
INVOICE				I
DELIVERY			I	D
ORDER	I	M	M	D

Figure 5.1

Matrix entries comprise single letters or combinations of the letters I,R,M,A,D. I indicates that an entity occurrence is inserted by the event, R indicates that an entity occurrence is read by the event, M indicates that an entity occurrence is modified by the event, A indicates that an entity occurrence is archived by the event and D indicates that an entity occurrence is deleted by the event.

The matrix provides a proving mechanism. It is easy to observe in figure 5.1 that the entity 'customer' needs events to insert and archive it, the 'product' entity needs maintenance events, the 'invoice' entity is inserted and never accessed and the 'delivery' entity is simply inserted then deleted. In general the solution to these anomalies is to introduce a number of new events onto the matrix and of course onto the entity event diagram. Reading across a row of the matrix gives the entity life history. The entity 'order' is inserted by event, 'order entry'; modified by events, 'produce order acknowledgement', and 'record delivery'; and eventually deleted by event, 'produce invoice'. Reading down a column gives the processing necessary. Event, 'record delivery', modifies the 'order' and 'product' entities and inserts the 'delivery' entity. The entity event matrix, therefore, provides a technique for plotting the effect of events on entities. It also provides a way forward into entity life histories.

5.3 Entity Life History

5.3.1 Introduction

Each row of the entity event matrix shows how an entity is effected by events. It does not show the sequence of events nor does it show when it is valid to carry out an event. The matrix only shows the normal events which effect the entity. The entity life history technique provides a graphical representation which shows the sequence of events and when it is acceptable to execute an event. Entity life histories also show abnormal events.

Figure 5.2 shows a simple entity life history with no abnormal events. Entity life histories have been modelled with place transition nets.

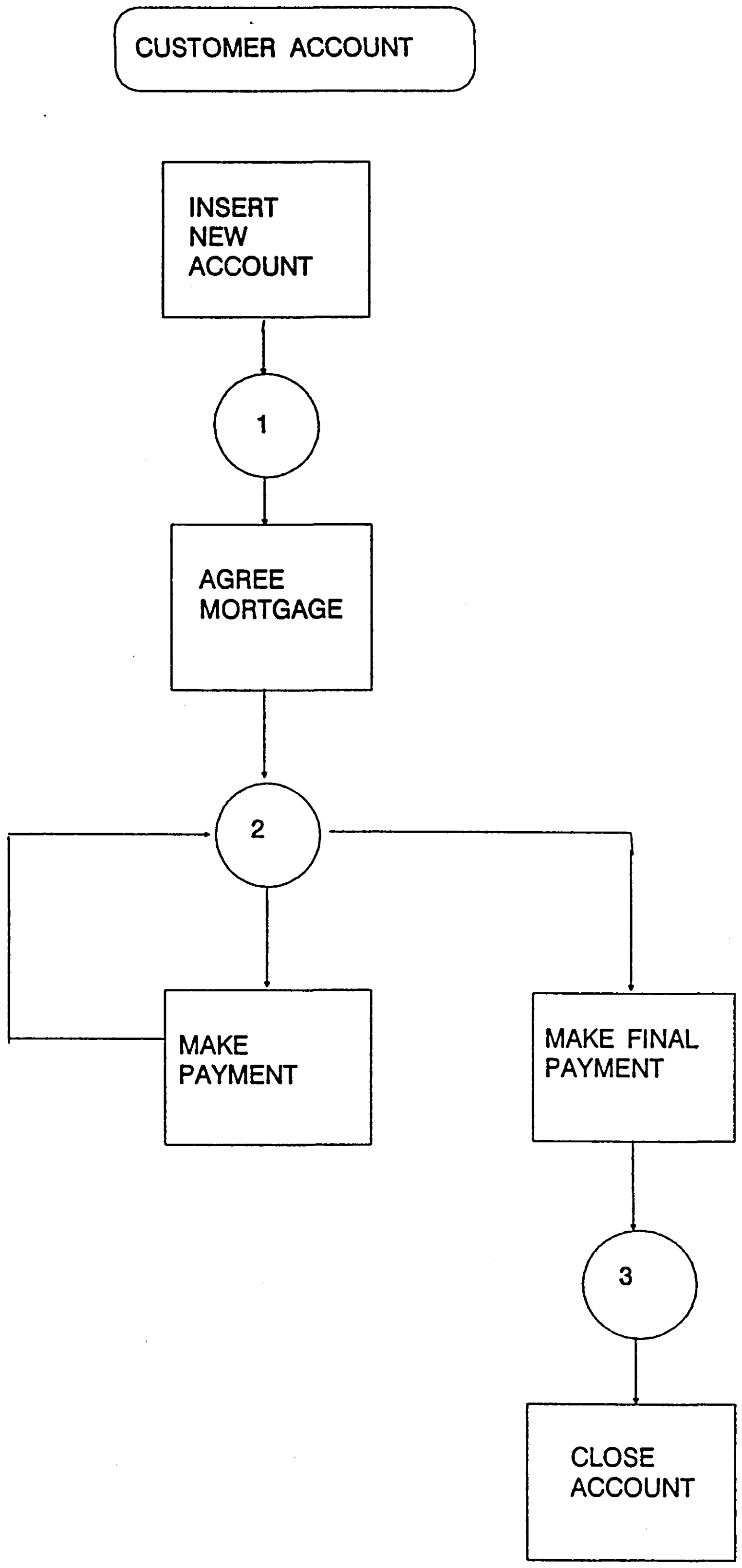


Figure 5.2

The rectangles represent events and the circles show the current status of the entity occurrence. The use of places to represent the entity statuses overcomes the problem of having to introduce statuses artificially onto the diagram after the diagram has been completed. The event, 'make payment', modifies the entity 'customer account'. It is only valid to make payments when the entity occurrence has a status of two. The status of the entity occurrence should be reset to two, in this instance, on completion of successful processing. The event 'make final payment', is only valid if the status is two; the set to status, in this case is three. The only valid event when the status is three, is 'close account'. Each event possesses, for each entity it effects, a valid previous status and a set to status.

Entity life histories allow the analyst to concentrate on a particular entity. In this case, perhaps the entity life histories should be amended to allow subsequent mortgage agreements. Figure 5.3 shows an amended entity life history for the entity, customer account.

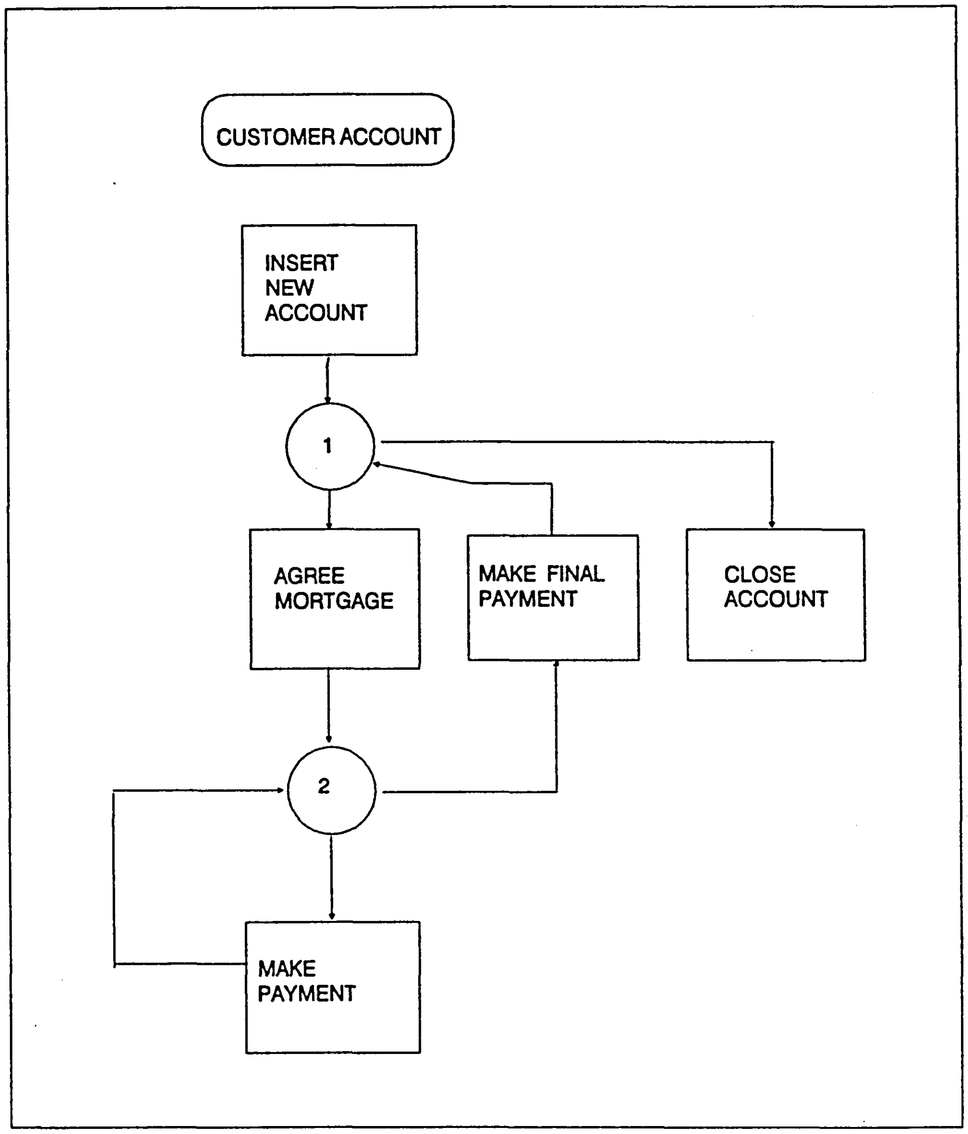

CUSTOMER ACCOUNT
INSERT NEW ACCOUNT
1
AGREE MORTGAGE
MAKE FINAL PAYMENT
CLOSE ACCOUNT
2
MAKE PAYMENT

Figure 5.3

This third view of the information system allows the following types of question to be answered:

1. Does each entity have an event which inserts it and an event which deletes it?

2. Is each entity accessed after insertion?

3. Which entities are referenced by an event?

4. Does each entity possess a complete life?

5. Does each event behave properly?

In this way the approach is able to validate the specification alongside its development ensuring correctness from within the approach.

5.3.2 Using Place Transition Nets to Model Entity Life Histories

For the purpose of entity life histories place capacities are always one, arc weights are one and the initial marking is such that the only enabled transition is the insert of an entity occurrence. Petri nets have been introduced into the development cycle by a software house, a pharmaceutical manufacturer and a major international bank after detailed discussions and consultancy work by the author. In all cases the results have exceeded expectations and have been very encouraging.

The development of entity life histories can be undertaken in a similar fashion to structured programming in that Petri net constructs exist which are similar to those used in structured programming. Petri nets support two further very important constructs, parallel and alternate. Each construct is discussed below.

5.3.3 Petri Net Constructs

Concurrency Structure. This is the most natural structure for Petri nets. Consider the entity life history, figure 5.4.

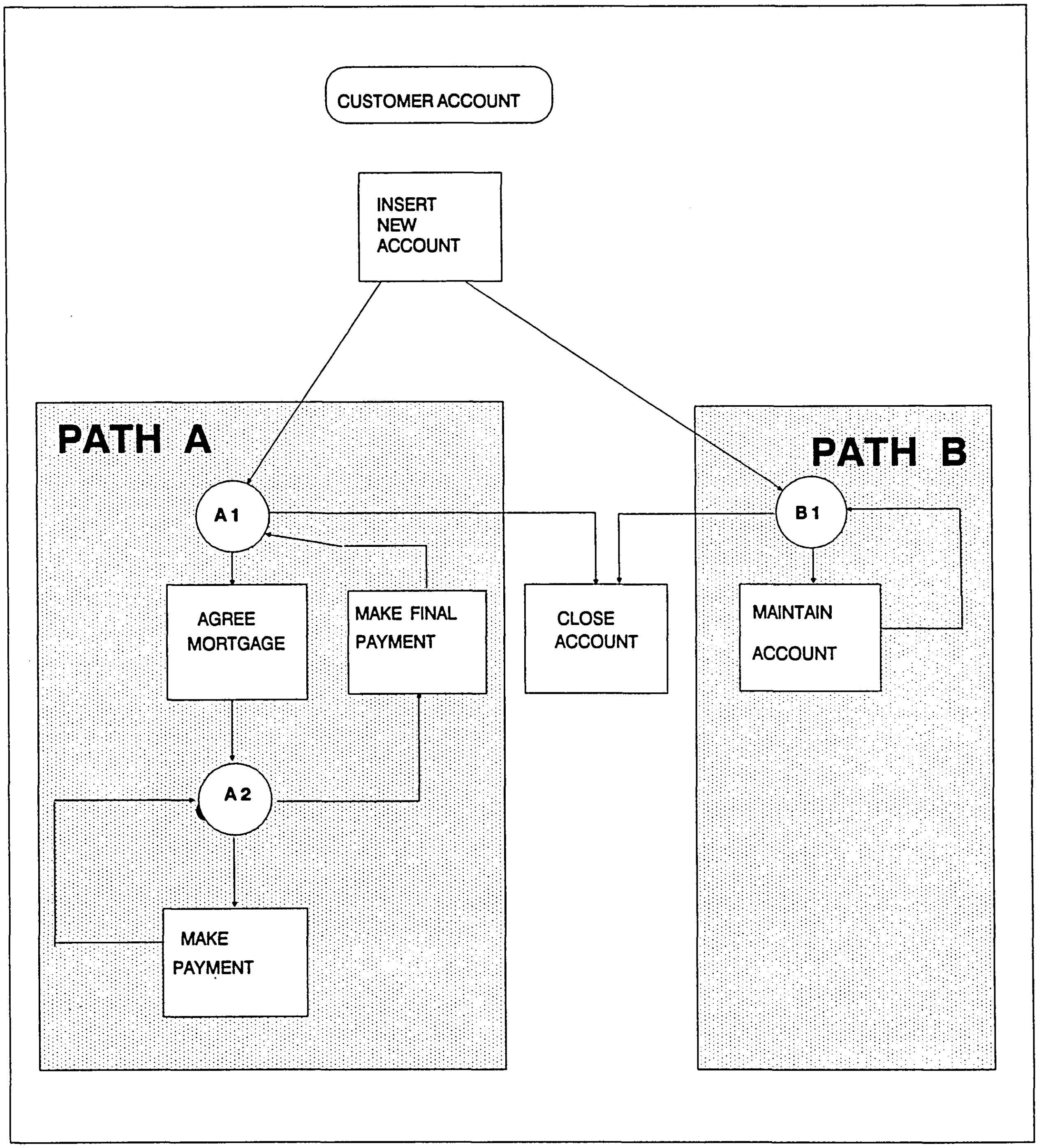


Figure 5.4

After an occurrence of the entity has been inserted into the database two concurrent paths are enabled. Path A shows the application of transactions to the entity whilst path B allows for the entity to be maintained at any time. Path A represents the agreement of a loan with a customer with subsequent payments and path B represents the change of address of the customer at any time during the loan repayment cycle. There are many examples where concurrent sets of activities are valid. The Petri net construct shown in figure 5.5 allows for both the start of concurrent processing of an entity occurrence and the conclusion of concurrent activity.

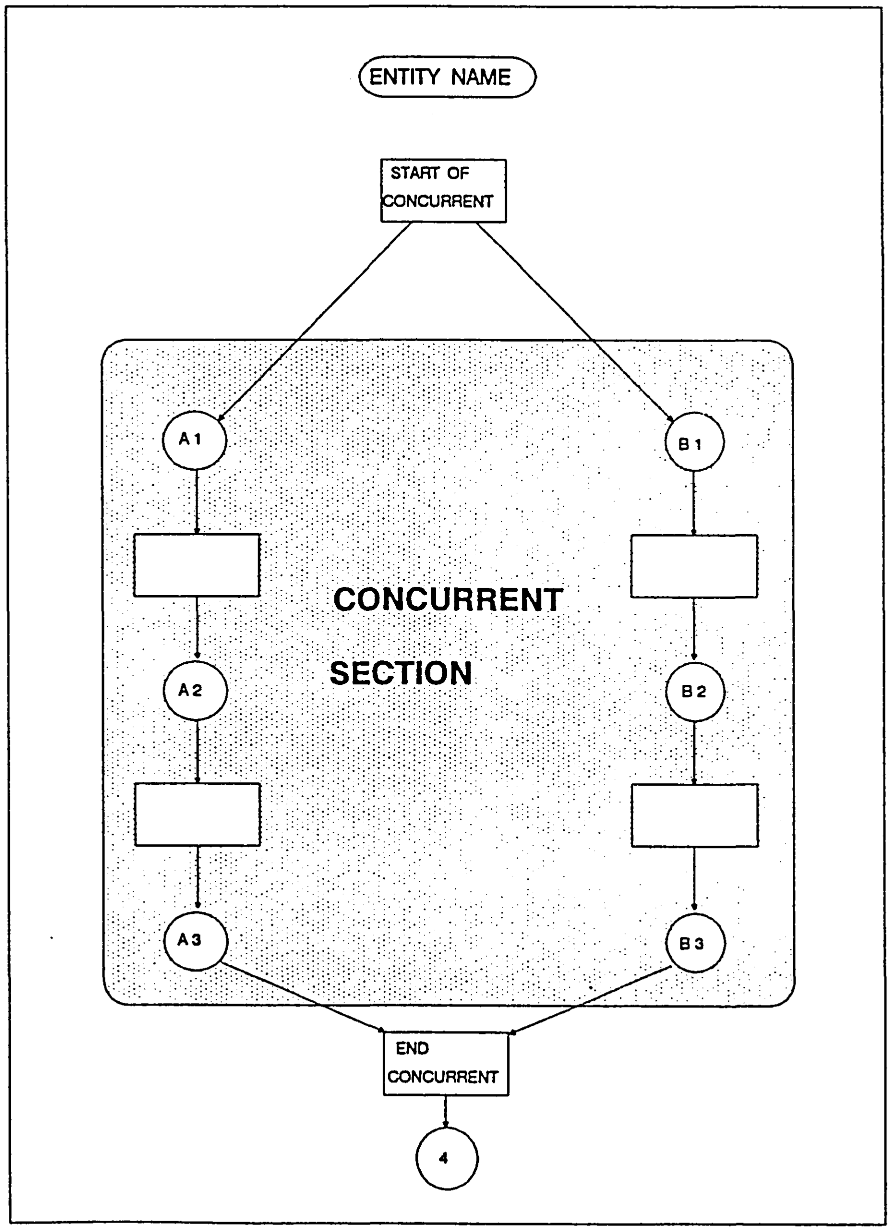


Figure 5.5

This construct overcomes the major problems experienced with the hierarchical and network representations.

Sequential Structure. Many processing tasks must be executed in sequence. Petri nets represent sequence very easily using the construct shown in figure 5.6.

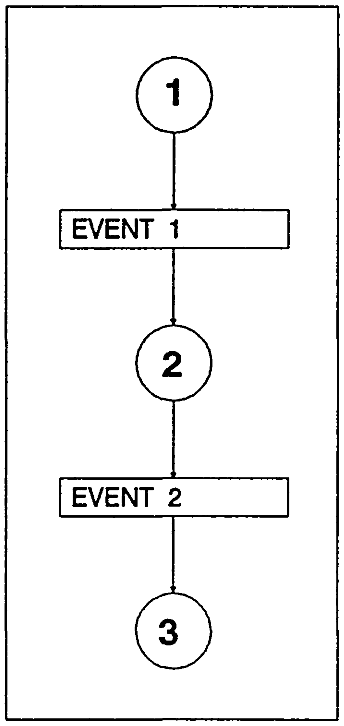


Figure 5.6

Selection Structure. Selection can be made on the basis of the result of a conditional expression, for example on the evaluation of the expression

customer current overdraft + new request > overdraft limit.

Depending upon the outcome of the expression, which can be immediately evaluated, a choice of processing can be expressed. Very often choice of processing comes from the need to be able to process error situations such as the demise of the customer in figure 5.7.

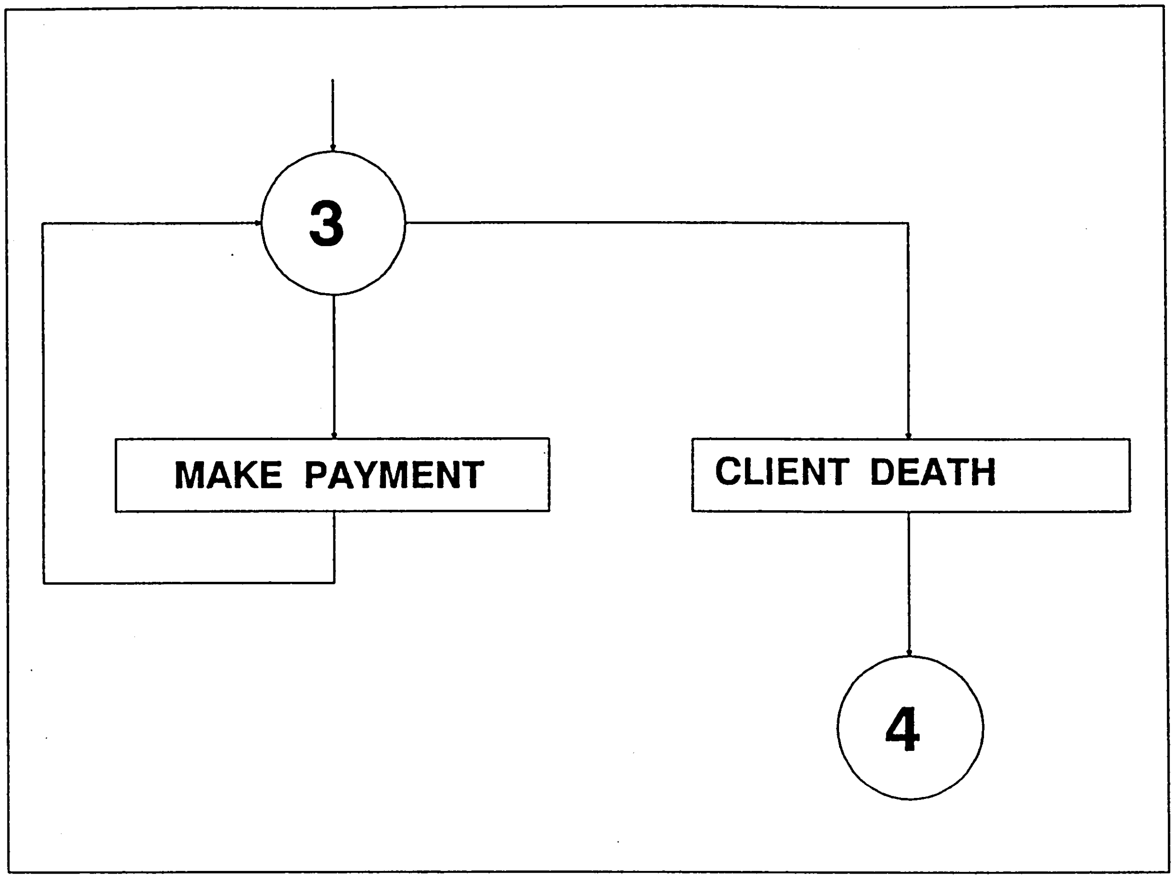


Figure 5.7

Petri nets use the following construct for choice, figure 5.8.

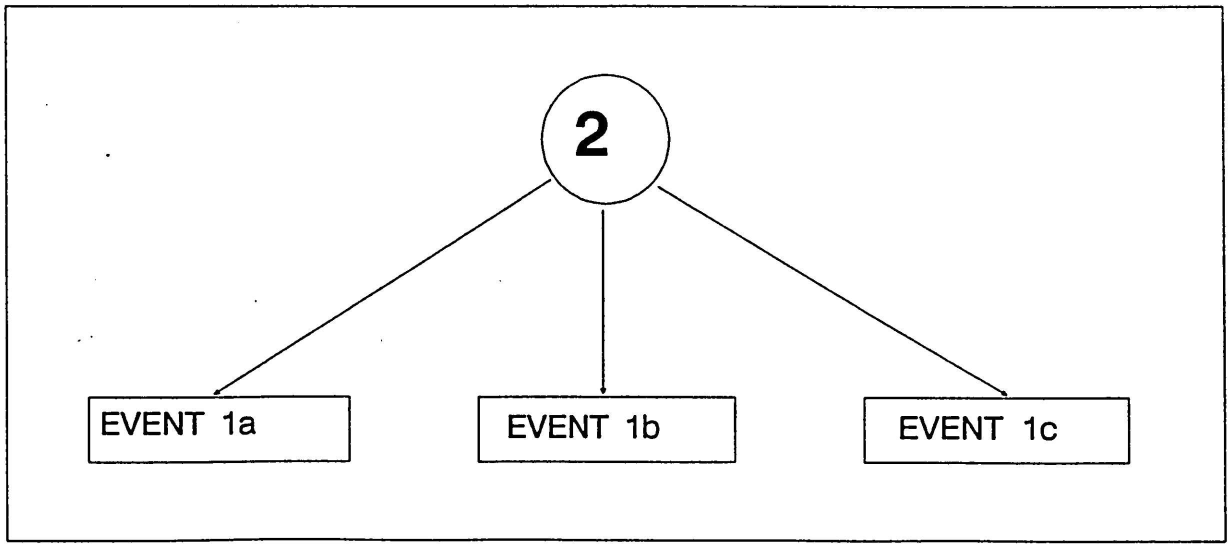


Figure 5.8

This is the natural way to represent choice. The hierarchical and network representations have to resort to optional and null events to represent choice.

Selection can also be influenced by outside interaction. Figure 5.9 shows that the entity application (form) may await a decision before further processing is possible. In this case the decision will effect the future processing.

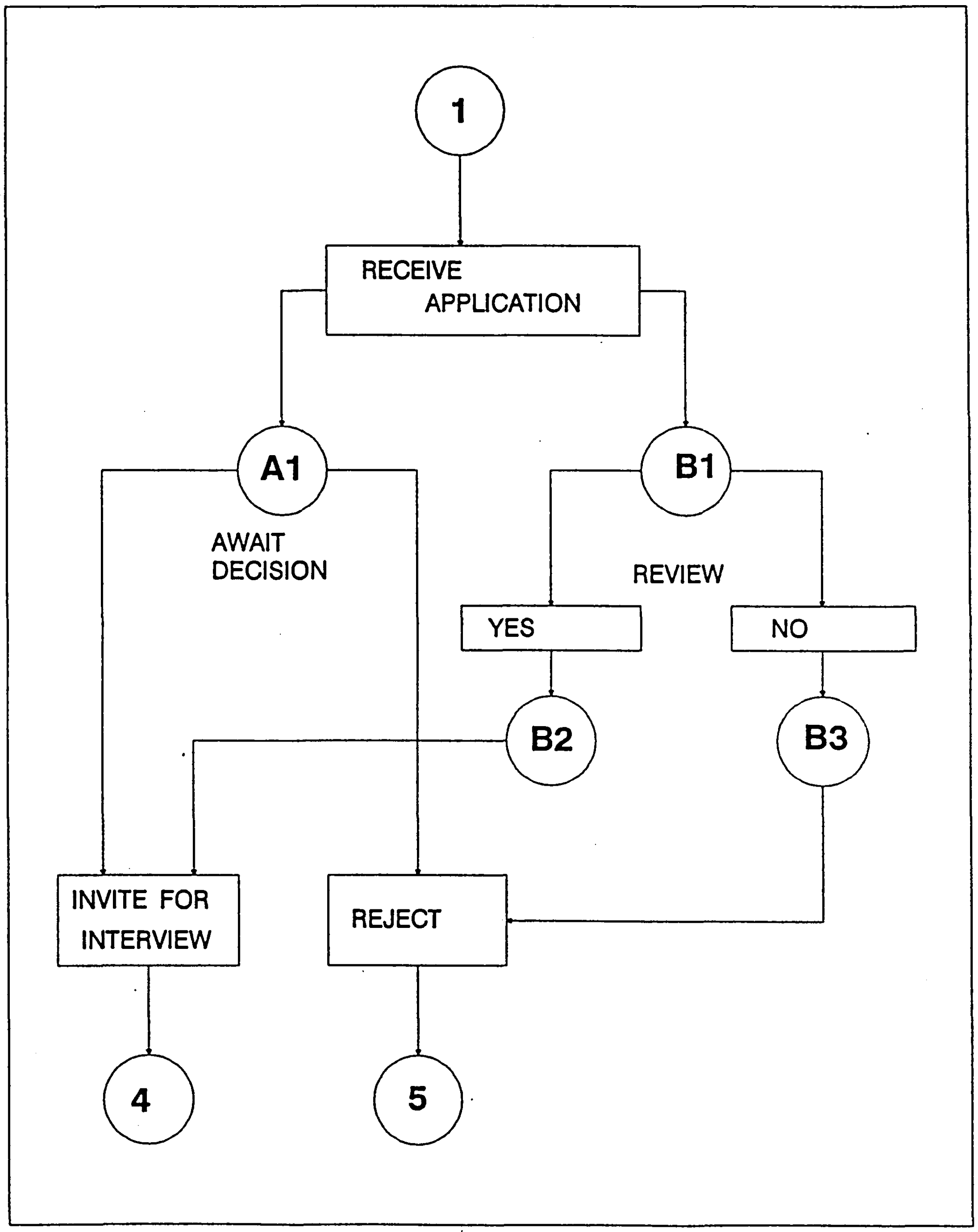


Figure 5.9

Alternate, A1, is different to choice, B1. This difference is important when one considers appropriate implementations for each construct. The Petri net construct for representing choice based upon an outside influence is shown in figure 5.10, it is an alternative structure.

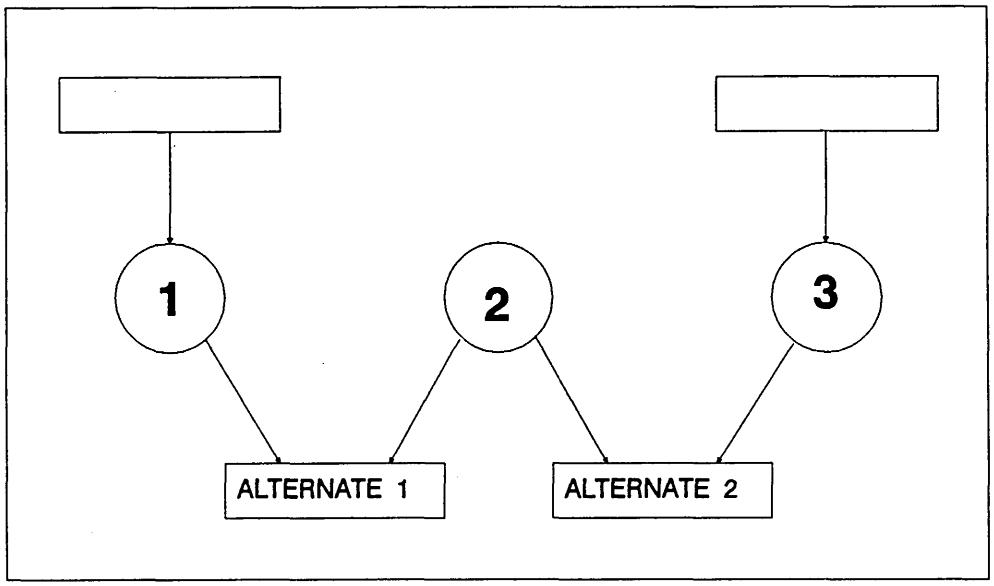


Figure 5.10

Iteration Structure. Iteration is very often required in the building of entity life histories, for example multiple repayments and the multiple loan pay cycle as shown in figure 5.3. The Petri net construct is simple, choice, together with sequence and parallel constructs, there is no need to introduce a new icon for iteration.

These constructs allow entity life histories to be constructed in a top down structured fashion.

5.3.4 Analysis of Entity Life Histories

A widely developed formal theory, based on linear algebraic techniques, exists for analysing basic Petri net properties which can be qualitative or quantitative. Qualitative analysis concerns net properties such as the absence of deadlock, liveness, boundedness, mutual exclusion and conflicts whilst quantitative analysis allows the evaluation of the dynamic behaviour and of the performance of the modelled system. Petri nets may be analysed using a variety of techniques including 'the token game', reachability trees and the use of invariants.

The paths through an entity life history represent possible entity lives from birth (insertion) to death (deletion). These paths can be traced by playing 'the token game'. The 'token game' is played by placing a token on the inaugural place, figure 5.11, and then following the enabling and occurrence rules of Petri nets.

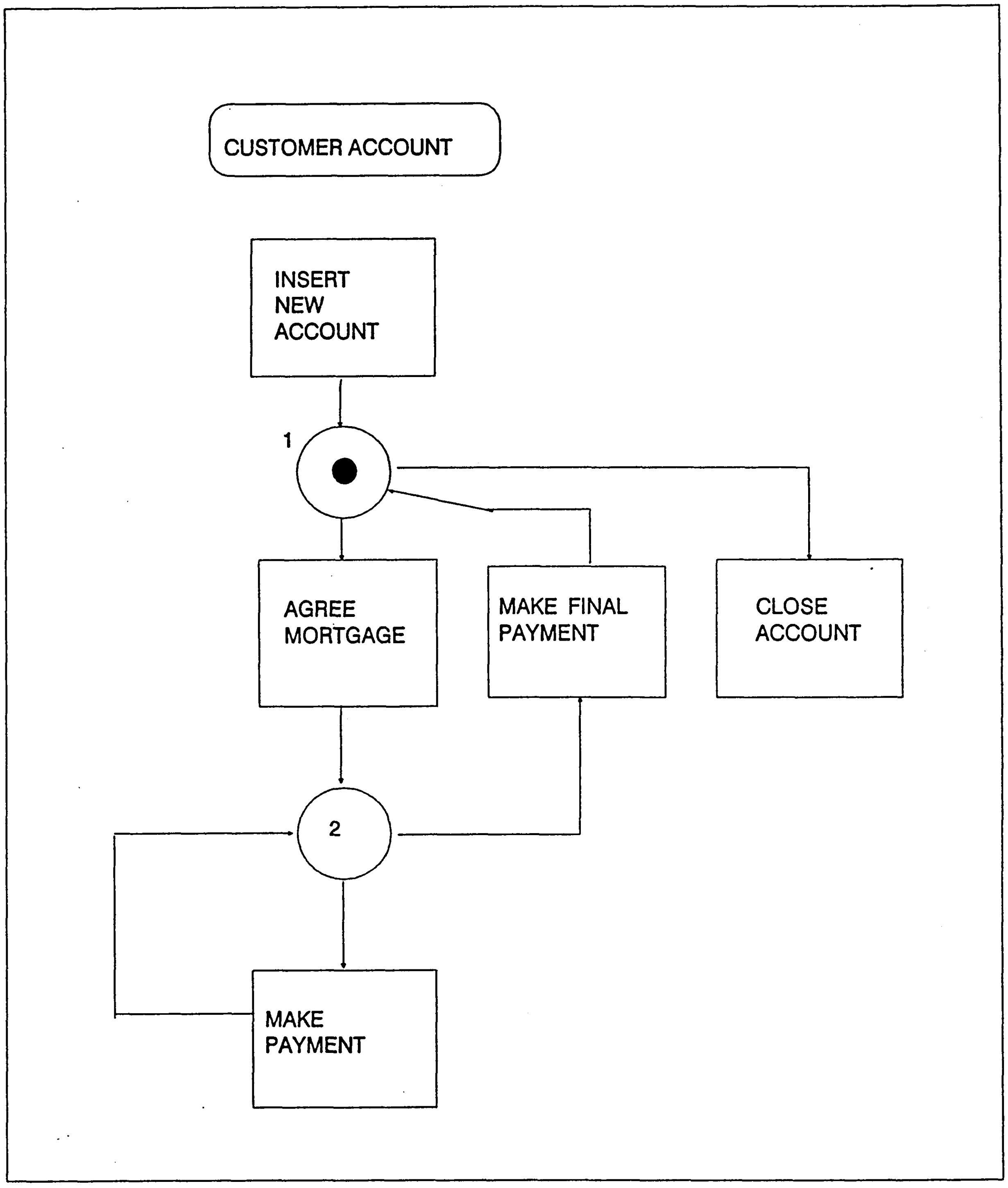

CUSTOMER ACCOUNT
INSERT NEW ACCOUNT
1
AGREE MORTGAGE
MAKE FINAL PAYMENT
CLOSE ACCOUNT
2
MAKE PAYMENT

Figure 5.11

When a transition is enabled, the corresponding 'move in the token game' may take place, we then say the transition or event occurs. The effect of an occurrence is that tokens are removed from the input places and added to the output places. Figure 5.12, 5.13, 5.14, 5.15 show the path of a customer account after insertion to deletion having behaved normally, that is negotiated an agreement for a mortgage, taken out a mortgage, paid it back, not taken out a second mortgage and therefore the account is deleted.

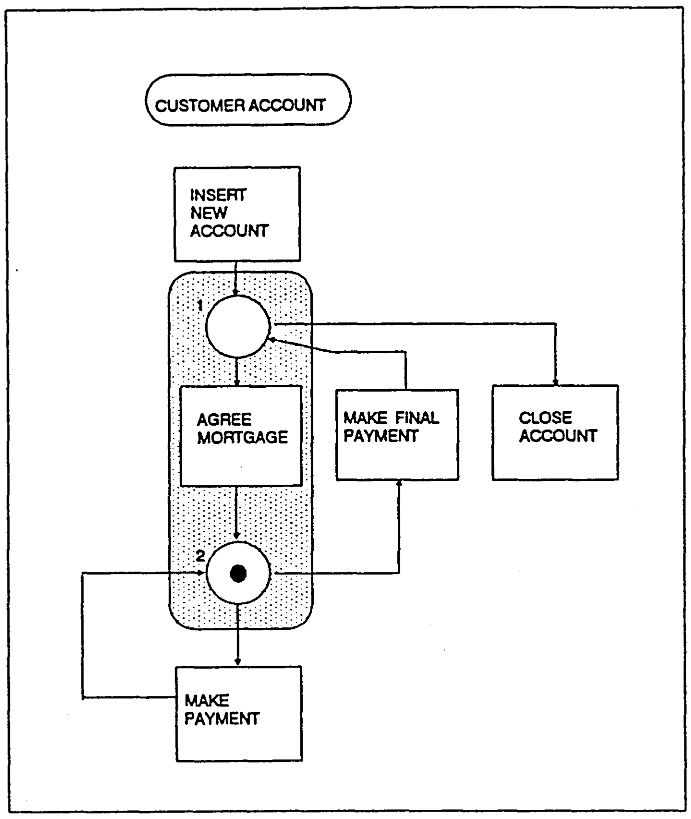


Figure 5.12

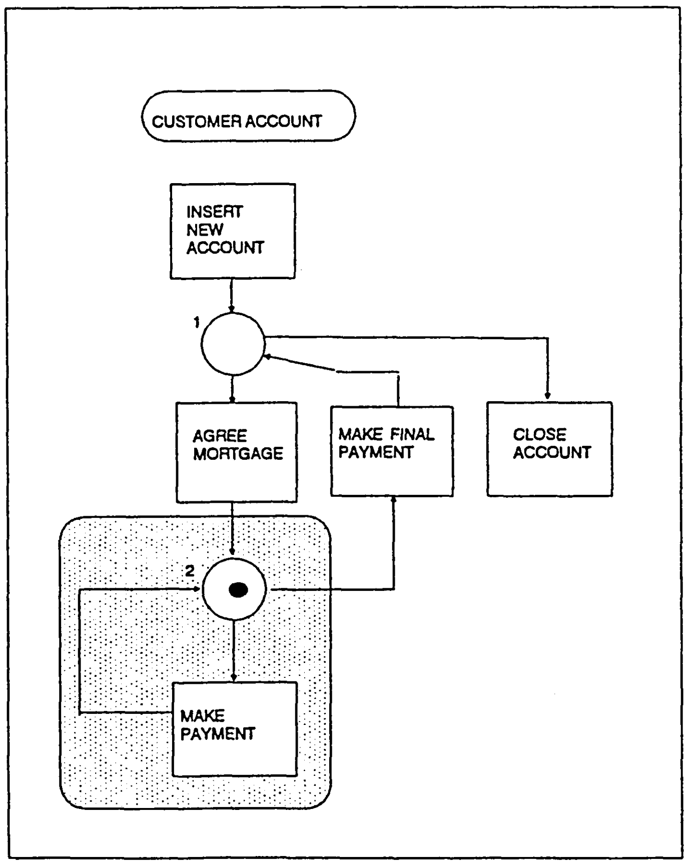


Figure 5.13

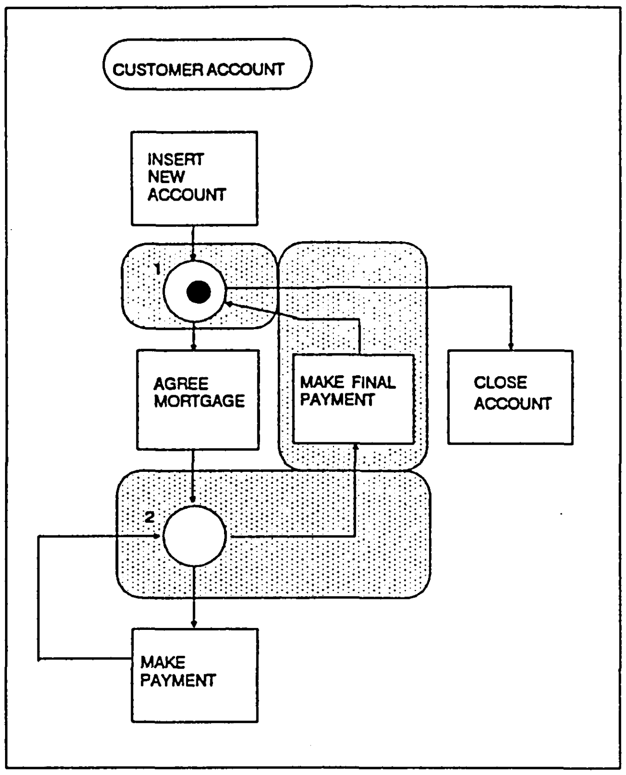


Figure 5.14

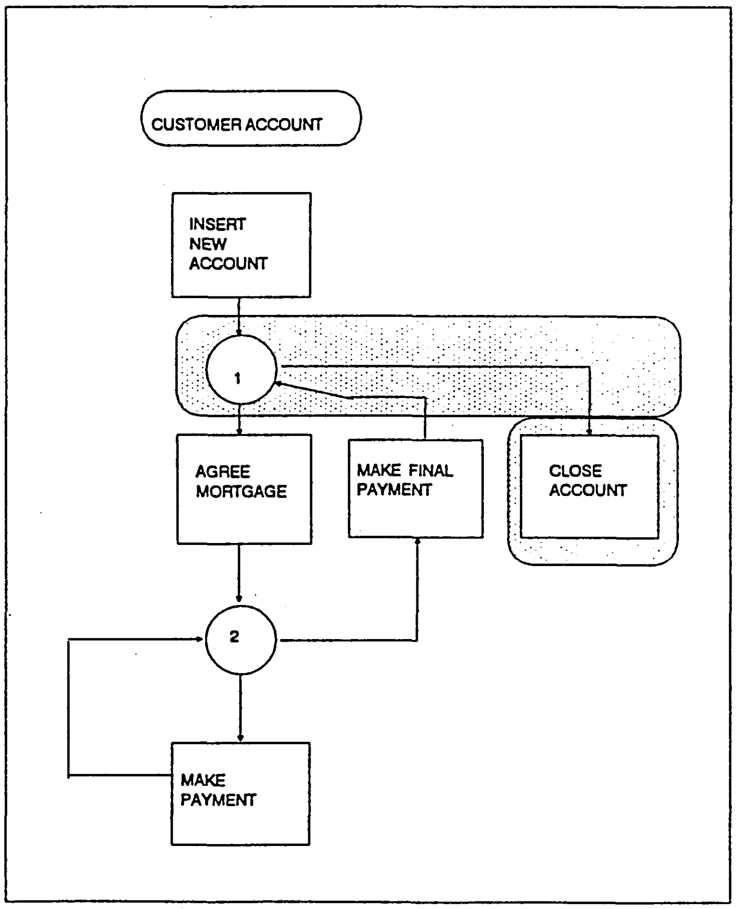


Figure 5.15

The 'token game' can be used further to investigate all possible routes to deletion by following all the different choice options available. In this way all of the possible lives for an entity may be established. The 'token game' is not guaranteed to discover all possible lives. This can be done with reachability trees.

The analysis of entity life histories by reachability tree assists the development of abnormal lives and the handling of error situations as well as leading to the discovery of all possible lives. Consider the entity life history shown in figure 5.16.

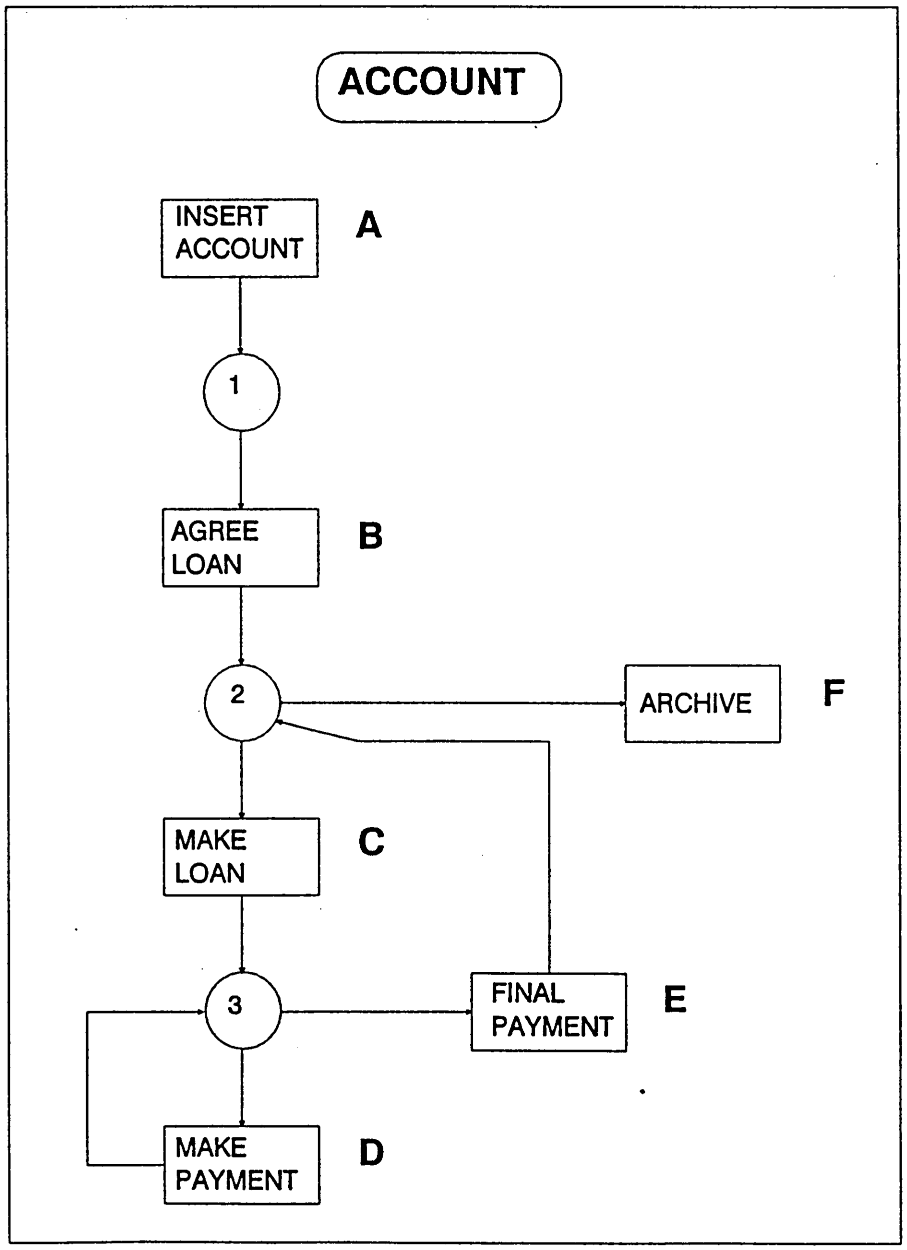


Figure 5.16

After a customer account is inserted no further events are allowed until the amount of a loan is agreed. Once agreement has been reached a loan can be made followed by zero or many payments and a final payment. From this status a further loan can be taken out or the account record can be archived. After insertion a token is deposited on place 1, there are zero tokens on all other places. The entity life history has a marking of (1,0,0). According to the enabling rules of Petri nets, the only enabled transition is 'agree loan'. If a loan is agreed then the transition 'agree loan' occurs and the marking becomes (0,1,0). From this marking either a loan can be made or the account record can be archived. If a loan is taken out the transition 'make loan' occurs and the marking becomes (0,0,1). From this new marking either a payment can be made or a final payment can be made, the new marking being (0,0,1) or (0,1,0) respectively. From marking (0,1,0) via the transition 'archive' we reach deadlock. Figure 5.17 shows the reachability tree.

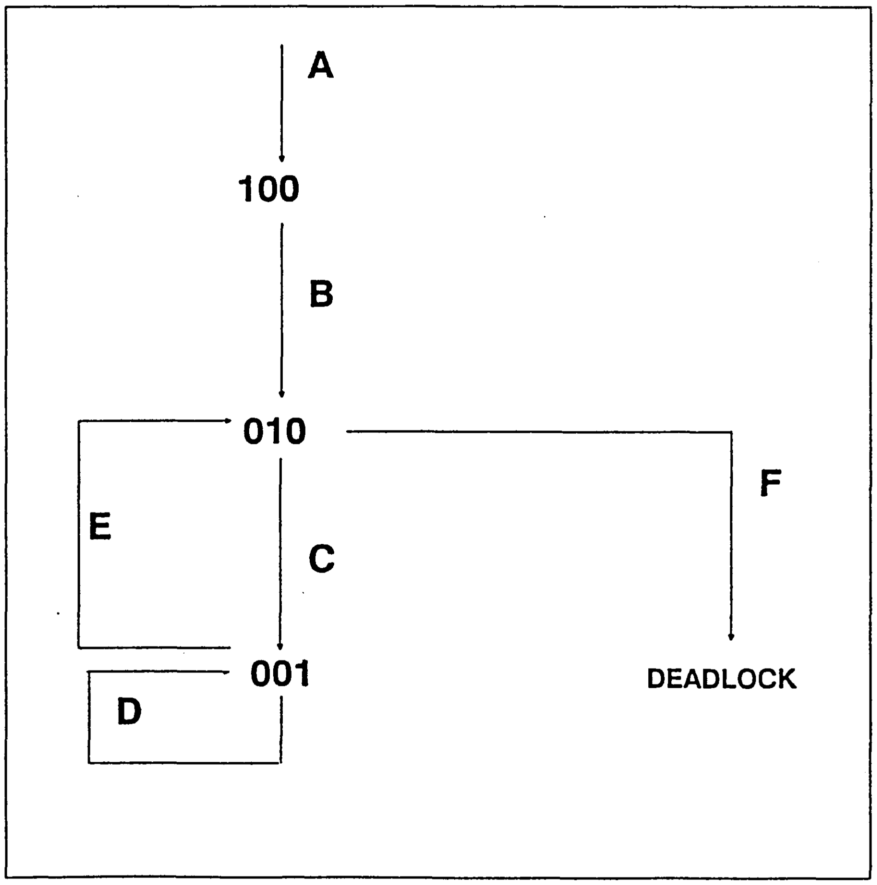


Figure 5.17

The reachability tree shows that an occurrence of the entity account is inserted (born), it has a proper life, and it is eventually archived (death) from which no further events are possible. Figure 5.18 shows an entity life history and figure 5.19 its corresponding reachability tree, note that there exists a possibility of deadlock. This entity life history is therefore not valid.

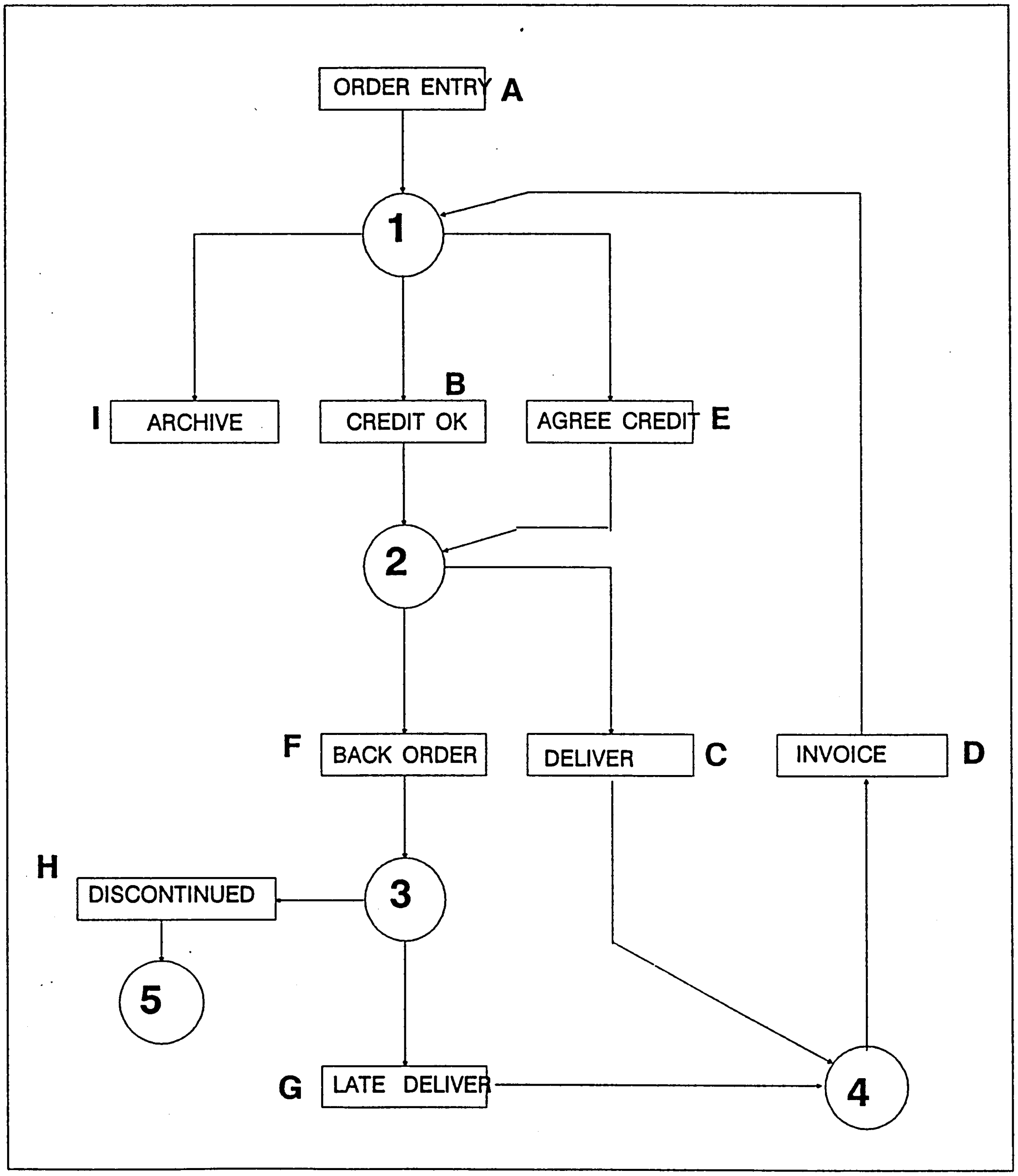


Figure 5.18

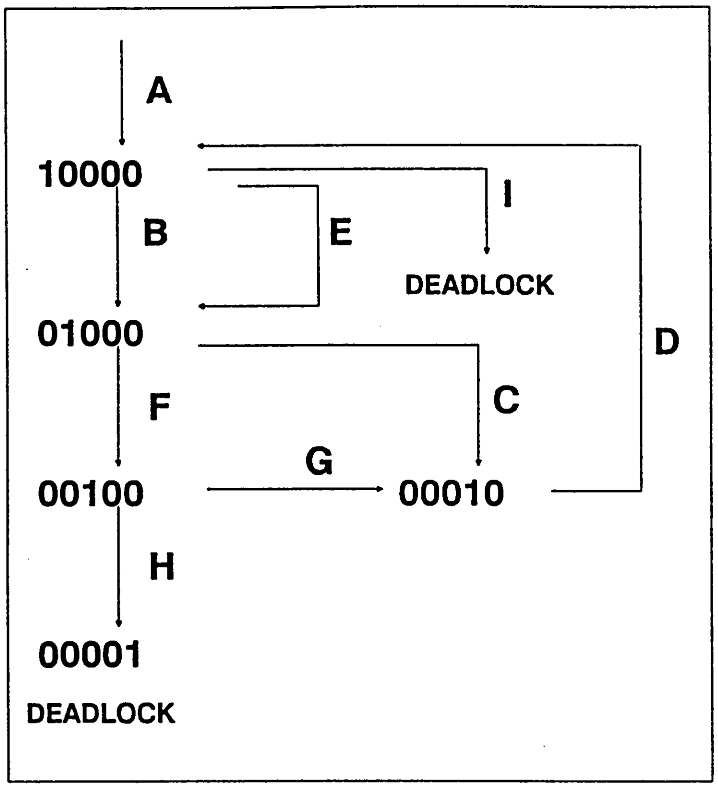
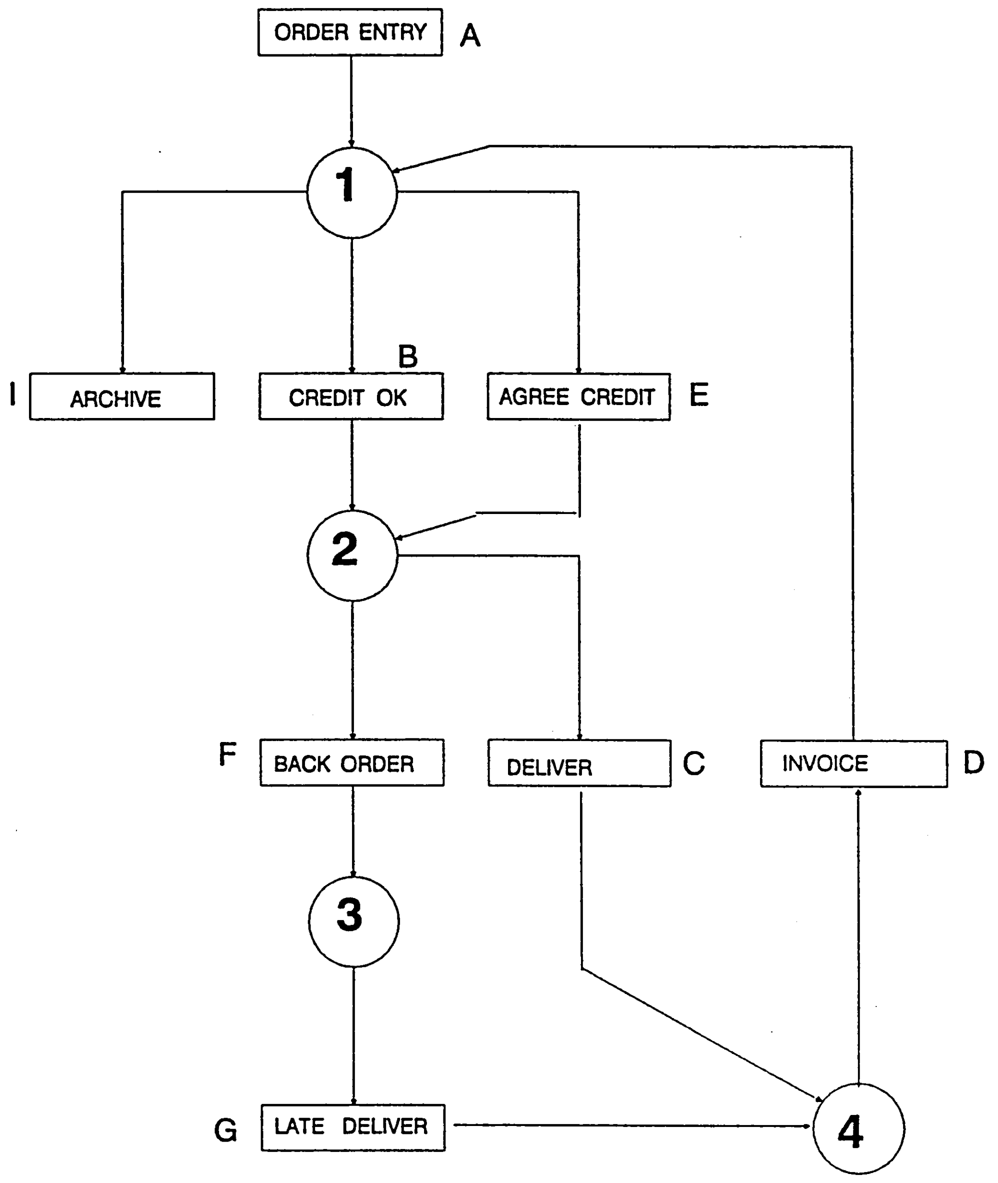


Figure 5.19

In fact two markings lead to deadlock. The marking following event I is the expected deadlock required after an order has been entered, cleared for credit, delivered and invoiced or entered, cleared for credit, back ordered, late delivered and invoiced. Note that credit clearance is available by two routes. However, if a order is back ordered and subsequently the stock item ordered is discontinued then a deadlock ensues leaving the order in limbo. Further action is required to remove orders from this limbo state which is represented by the marking following event H.

The use of reachability trees is a more rigorous approach, however, as soon as the net reaches any significant size the tree becomes large and unmanageable since it charts all of the occurrence sequences. It is sometimes easier to analyse the underlying structure of the net instead of observing its behaviour. The use of invariants proved to be very useful in the discovery of net properties.

A Petri net and its incidence matrix are shown in figure 5.20.



	A	B	C	D	E	F	G	I
1	1	-1		1	-1			-1
2		1	-1		1	-1		
3						1	-1	
4			1	-1			1	

Figure 5.20

Transition 'A', the insertion of an order occurrence, is a special transition in that it creates tokens. Similarly transition 'I' is special in that it removes tokens. If we exclude transitions 'A' and 'I', once a token representing an order is inserted into the database then it remains until removed by transition 'I'. This can be shown by examining the matrix shown in figure 5.21 which excludes the insert and archive transitions and noting that a place invariant is I.

	B	C	D	E	F	G		I
1	-1		1	-1				1
2	1	-1		1	-1			1
3					1	-1		1
4		1	-1			1		1

Figure 5.21

The interpretation of this invariant is that

1 times the number of tokens on place 1

+ 1 times the number of tokens on place 2

+ 1 times the number of tokens on place 3

+ 1 times the number of tokens on place 4

= (the number of tokens introduced by transition 'A')

- (the number of tokens removed by transition 'I').

This proves that order entity occurrences, once entered into the database, remain in the database until they are archived. Place invariants provide valuable results concerning the structure of entity life histories particularly in the detection of deadlocks and traps.

$J1^T = (0, 1, 1, 1, 0, 0, 0, 0)$,

$J2^T = (0, 0, 1, 1, 1, 0, 0, 0)$,

$J3^T = (0, 1, 0, 1, 0, 1, 1, 0)$,

$J4^T = (0, 0, 0, 1, 1, 1, 1, 0)$

are all transition invariants of N, figure 5.20, in that they satisfy $N.J = 0$.

If we take $M_0 = (1, 0, 0, 0)$ which represents an order newly inserted into the database then we can observe that this marking is reproducible, enabling the order to be archived, under any if the transition invariants above.

J1 represents credit OK, deliver, invoice;

J2 represents agree credit, deliver, invoice;

J3 represents credit OK, back order, late deliver, invoice; and

J4 represents agree credit, back order, late deliver, invoice.

Transition invariants show all of the valid routes through an entity life history. This information is very useful in ensuring the specification meets the user requirements and in the construction of test specifications.

We have seen how deadlock can be observed using reachability trees. We have also shown that place invariants provide valuable information about the number of tokens on a Petri net. We will now use these ideas to check the structural properties of entity life histories. Examine the set of places (1, 2, 3, 4) in figure 6.20. Note that with transition 'A' excluded, if the set of places becomes empty it remains empty. This can be shown by asserting that each transition which inserts a token into the set (1, 2, 3, 4) requires a token from the set to occur. Software exists to detect such sets of places. This software was used successfully in the identification of a number of potential deadlocks in software for telephone exchanges.

Transition B inserts a token onto place 2 but requires a token on place 1 to occur;

transition E inserts a token onto place 2 but requires a token on place 1 to occur;

transition F inserts a token onto place 3 but requires a token on place 2 to occur;

transition C inserts a token onto place 4 but requires a token on place 2 to occur;

transition G inserts a token onto place 4 but requires a token on place 3 to occur; and

transition D inserts a token onto place 1 but requires a token on place 4 to occur.

We know that transition 'A', order entry, inserts a token representing an order occurrence into the set (1, 2, 3, 4). If it is removed from the set the set will become empty and deadlock will occur. P is called a deadlock iff the preset of P is a subset of the post set of P. Observe that transitions H and I remove a token from the set. Transition 'I' is acceptable, it is the archive process, transition 'H' is not acceptable as it leaves the token or order occurrence in limbo. The life history must be amended to overcome this problem, generally this means that some events have been overlooked.

Examine figure 5.22 and note that the set of places (3, 5) represents a trap. Q is called a trap if the post set of Q is a subset of the preset of Q. Once a token, entity occurrence is back ordered then it remains back ordered forever, that is it is trapped.

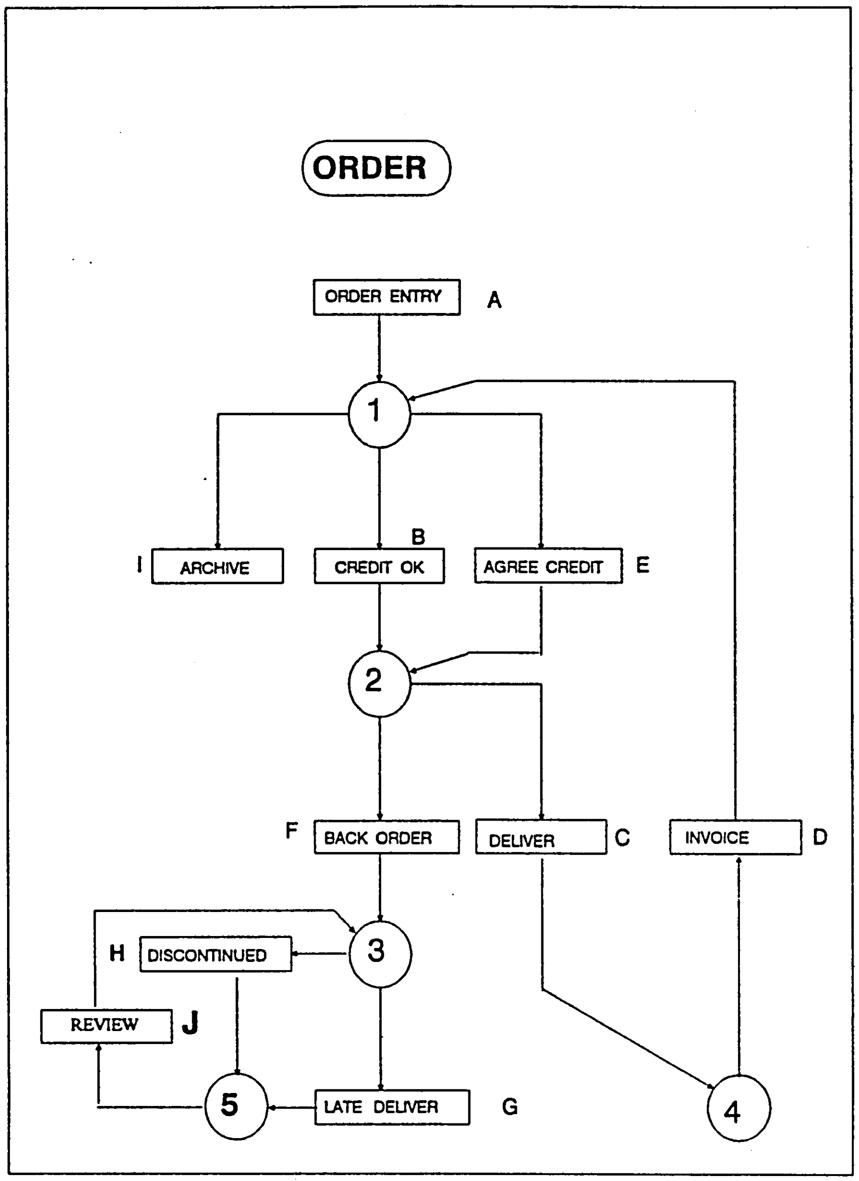

ORDER
ORDER ENTRY
A
1
I
ARCHIVE
B
CREDIT OK
AGREE CREDIT
E
2
F
BACK ORDER
DELIVER
C
INVOICE
D
3
H
DISCONTINUED
REVIEW
J
5
LATE DELIVER
G
4

Figure 5.22

It is possible to establish all the sets of places which represent deadlocks or traps, therefore it is possible to analyse the structure of the entity life history separately from its behaviour. Again the entity life history requires further development.

5.4 Conclusions

It has been shown that the use of place transition nets for entity life histories overcomes all of the drawbacks discusses earlier. Place transition nets have natural constructs for sequence, selection and iteration and more importantly also for parallel and alternate. Further the places naturally represent the entity status. Error and abnormal situations can be handled easily by the use of the choice construct. Exercises conducted with practising analysts and students have shown the token game to be very useful. It is possible to discover all error and abnormal processing, it is possible to ensure there is always a route to end of life and it is possible to produce complete specifications. The use of Petri nets for entity life histories has proved to be very successful.

Reachability trees and invariants were used during the case study work. They ensured that the entity life histories were complete and consistent. Analysis of entity life histories using trees and invariants has now been introduced into consultancy work and into teaching.

Chapter 6 DEVELOPMENT OF COLOURED PETRI NETS AND SYSTEM NETS

6.1 Transformation of Entity Event Diagrams and Entity Life Histories into Coloured Petri nets

6.1.1 Introduction

The approach now requires that a coloured Petri net be developed to provide a model of the complete information system. Coloured Petri nets can be very difficult to construct directly from a specification of requirements. The reasons for this were discussed earlier. Given below are a set of formal rules for transforming the entity event diagrams and the entity life histories into a coloured Petri net. These rules have been automated using the software discussed in chapter 7.

6.1.2 The Transformation Rules.

A set of formal transformation rules for EEDs and ELHs into a Coloured net that characterise semantic equivalence is given below. The list is divided into groups indicating the scope and purpose of the transformations.

The transformations are formulated in terms of changing a set of lowest level EEDs.

EED = (EEV,EEX,EEN,EDF,EG) where

EEV is the set of events

EEX is the set of external entities

EEN is the set of entities

EDF is the set of data flows

EG is the set of guards

$EEV \cap EEX \cap EEN = \emptyset$

EDF is a subset of $(EEV \times EEX) \cup (EEX \times EEV) \cup$

$(EEV \times EEN) \cup (EEN \times EEV)$

EG is a subset of (EEV x EEV') where EEV' not = EEV

and a set of Entity Life Histories

ELH = (HEV, HES, HEF) where

HEV is the set of events

HES is the set of entity states

HEF is the set of flows

$HEV \cap HES = 0$

HEF is a subset of $(HEV \times HES) \cup (HES \times HEV)$

into a coloured Petri net CPN = (P,T,A,S,C,V,G,E,M) where

1. P is a finite set of places, T a finite set of transitions and $X = P \cup T$ is a set of nodes.

2. A, a subset of $(P \times T \cup T \times P)$, is a set of arcs.

3. S is a finite set of types, called colour sets.

4. C is a colour function mapping from P into S. It attaches to each place a set of possible token colours.

5. V is a variable function defined from T into finite sets of variables, such that each variable belongs to S.

6. G is a guard function defined from T into predicates, such that G(t) has a

finite set of variables VAR(G(t)), a subset of V(t), and [Type(g(t) = Boolean and Type(Var(G(t))) sub set S] for all t∑T.

7. E is an expression function defined from A into expressions, such that [Type (E(a)) = C(p(a)) and Type (Var(E(a))) subset S] for all a∑A.

8. M is the initial marking.

Transformation of the Entity Life Histories.

1. Every event HEV of ELH becomes a transition T of CPN.

2. Every entity state HES of ELH becomes a place P of CPN.

3. Every flow HEF of ELH becomes a flow A of CPN.

(The ELH are simply collected onto one diagram)

Transformation of the Entity Event Diagrams.

1. Transformation of guards. (EEV x EEV')

a) Create a new place, P of CPN, name the place with the guard name.

b) Create an arc between EEV and the new place, where EEV = HEV (T of CPN).

c) Create an arc between the new place and EEV' where EEV' = HEV' (T of CPN).

2. Transformation of Entity Reads. (EEN x EEV).

a) Create a reference place, P of CPN, name the place with the entity name.

b) Create a double arc between the new reference place and EEV = HEV (T of CPN).

These rules give the places, transitions and arcs for the Coloured Petri net.

Colour Sets and Colour Functions.

The set of colour sets is created by the union of the attributes from the entity tuples and the attributes from the guard tuples. The colour function maps the entity tuples onto coloured places from the corresponding entity life histories and the guard tuples onto each place generated from a guard.

Variables.

The variables for each transition comprise all the colour sets from the transition's surrounding places together with any colour set read by the transition.

Guard Functions and Expression Functions.

Guard functions are determined from the event descriptions. They represent any condition which may prevent the event from occurring. Expression functions reflect the event functionality.

Initial Marking.

The initial marking is given by the distribution of entity tuples in the database.

To illustrate the transformation from ELHs and EEDs to a Coloured Petri net first consider figure 6.1. This figure shows extracts from four entity life histories, order number, a simple parameter, order head, order line and stock.

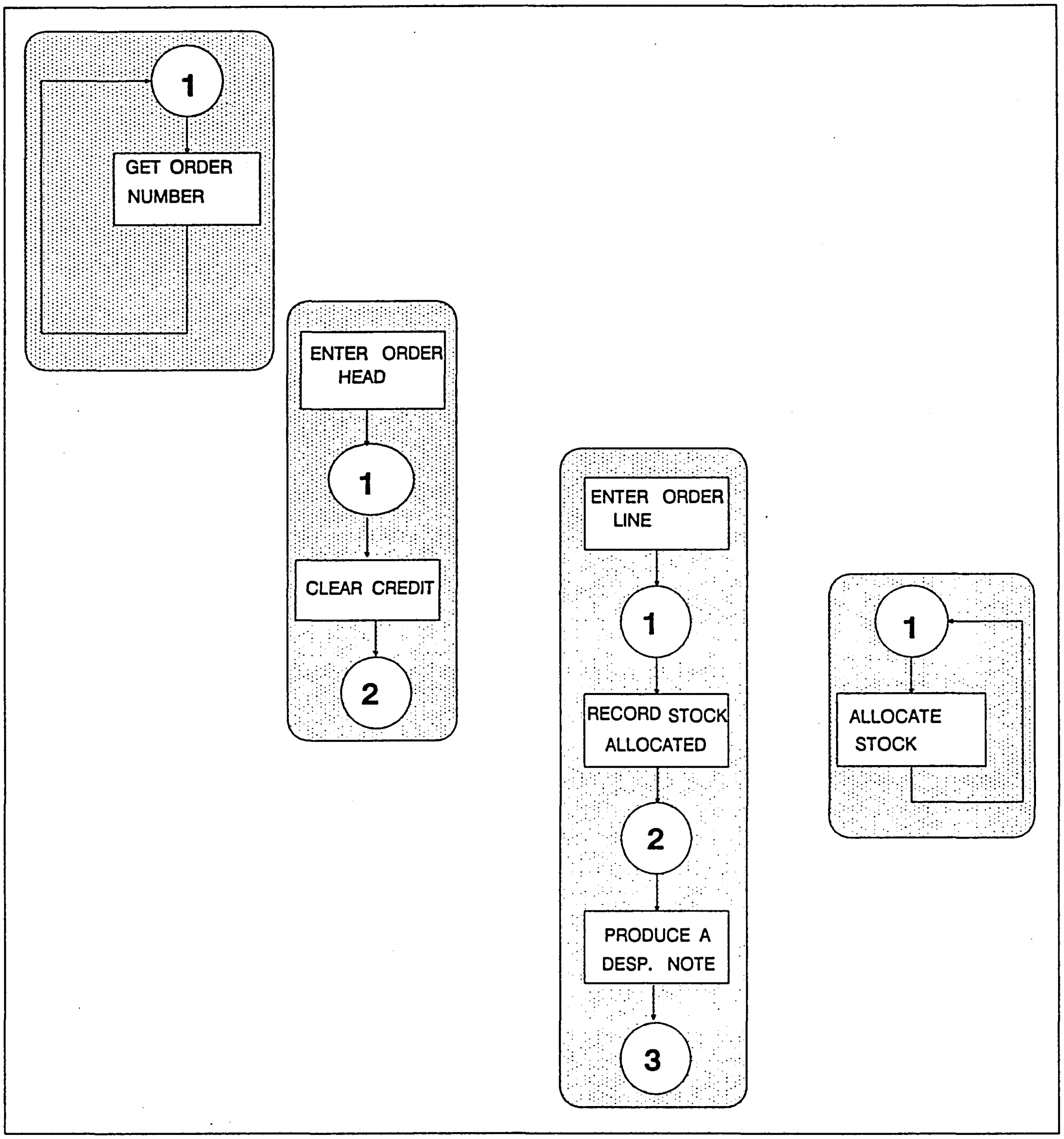

1
GET ORDER NUMBER
ENTER ORDER HEAD
1
CLEAR CREDIT
2
ENTER ORDER LINE
1
RECORD STOCK ALLOCATED
2
PRODUCE A DESP. NOTE
3
1
ALLOCATE STOCK

Figure 6.1

Figure 6.2 shows an entity event diagram.

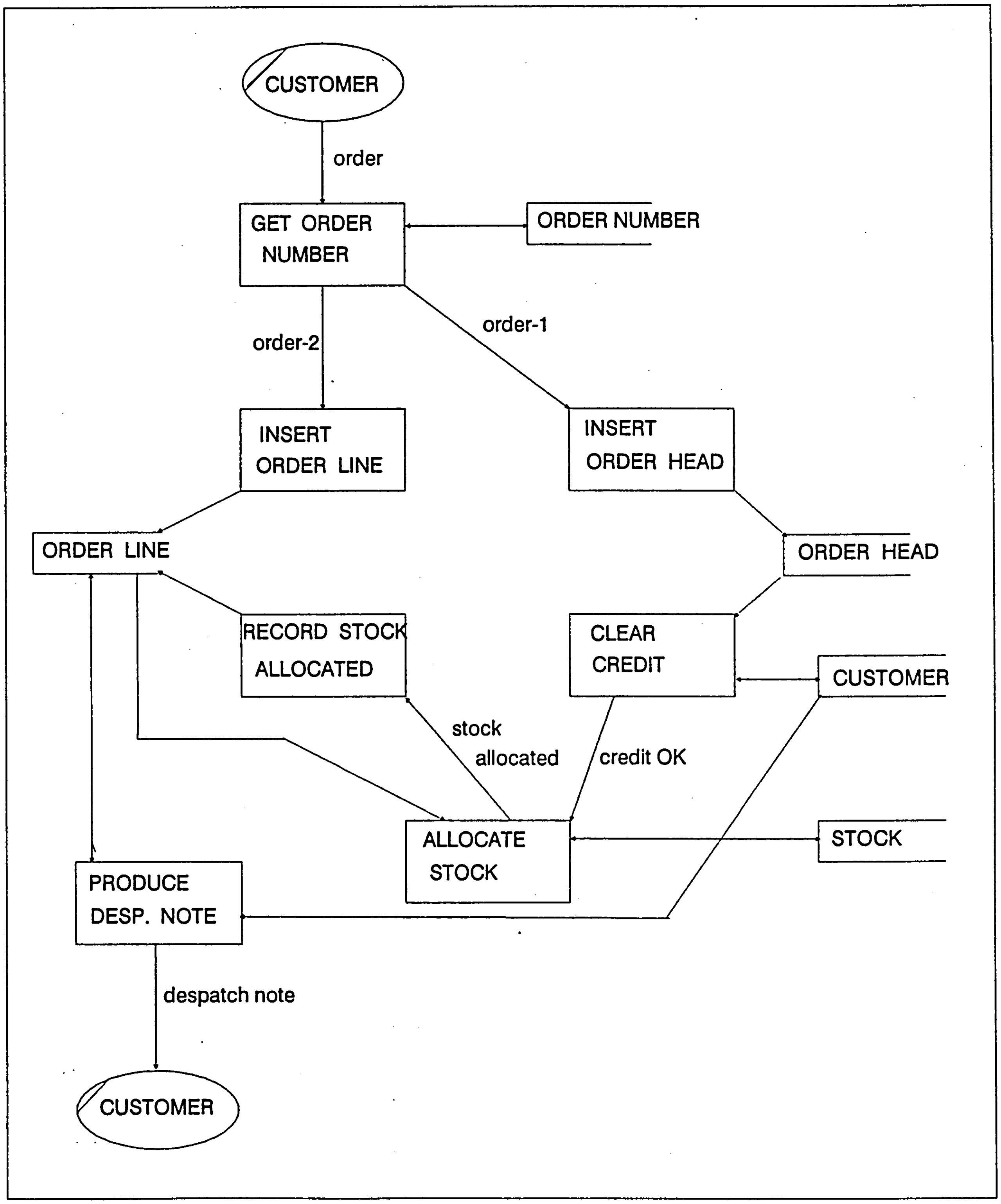


Figure 6.2

Orders are inserted as an order head and multiple order lines after an order number has been obtained. The insertion of order head and order lines is guarded by the event get order number. The order head is cleared for credit. The allocation of stock is guarded by the credit clearance event and the recording of any stock allocated on the order line is guarded by the allocate stock event.

Figure 6.3 shows the four new places representing the guards.

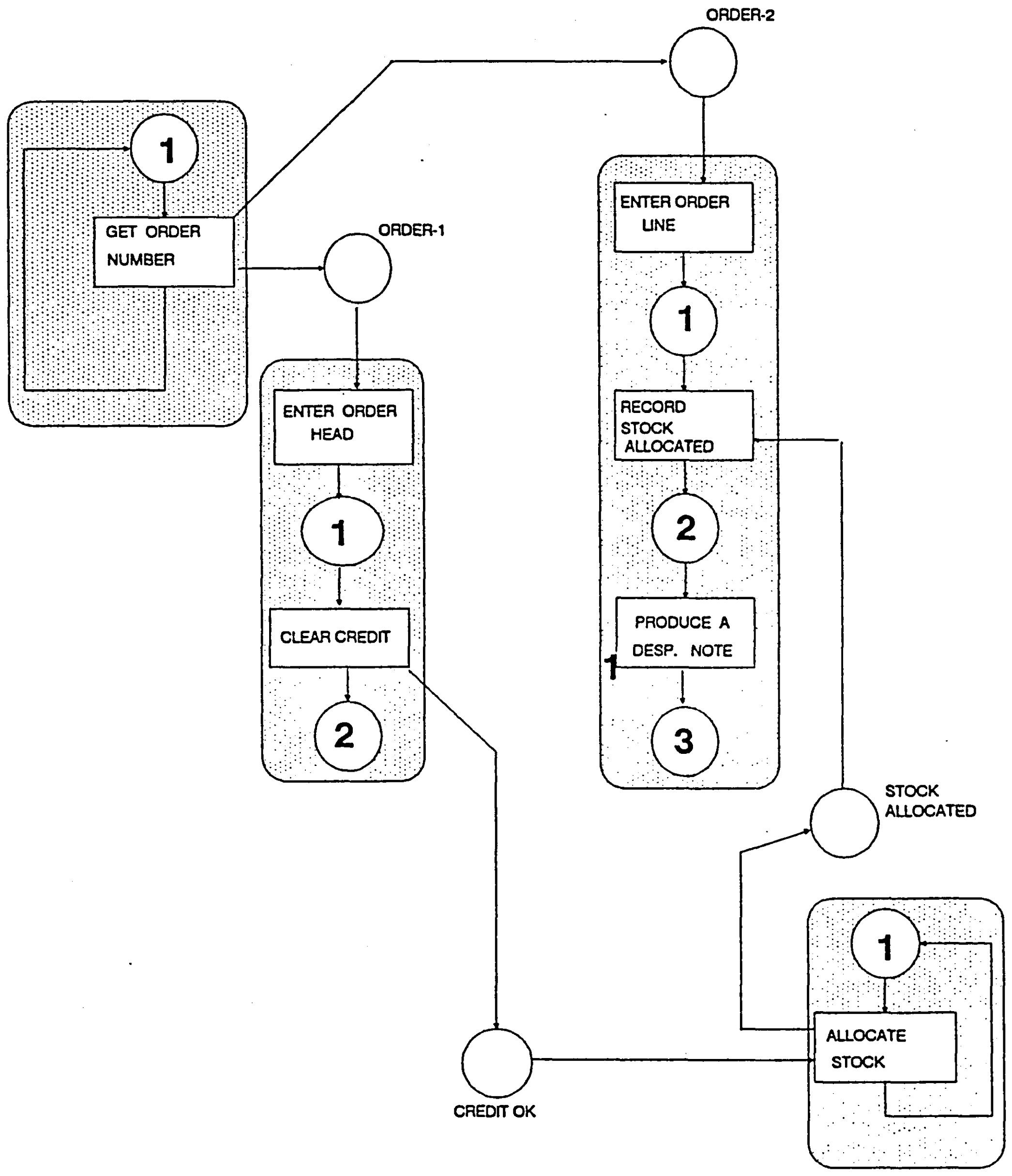


Figure 6.3

Figure 6.4 shows the places, transition and arcs for the Coloured Petri net.

Colour Sets

order_no	available order numbers
cust_no	customer numbers
no_of_lines	the number of lines on an order
product_no	product numbers
qty_ordered	the quantity ordered on an order line
qty_allocated	the quantity of stock allocated to an order line
qty_in_stock	the quantity of a product in stock
stopped	the customer status, stopped yes/no

Colour Function

S1 = order_no;

S2 = product cust_no * order_no;

S3 = product order_no * product_no * qty;

S4 = product product_no * qty_in_stock;

S5 = product order_no;

S6 = product order_no * product_no * qty_allocated;

Variables

var o,o':order_no;

var c,c':cust_no;

var s:stopped;

var p,p':product_no;

var q,q':qty;

Guard Functions

Guard functions are required where more than one colour set provides the input. A guard is required to match the tuples from each colour set. For example, to allocate stock the order from the credit OK must find all its order lines (o = o') and the product from the order line must be a valid product (p = p').

The event functionality can influence the guard. To clear credit the order head must locate its customer tuple and the stopped status must be 'no'.

Expression Functions

The expression functions for allocate stock are min. and max. The stock required is represented by q', stock on hand by q. The maximum of 0 and q - q' is returned to stock. If $q >= q'$ then the stock remaining is returned to stock else if $q < q'$ then q - q' is negative and the stock level is set to zero. The stock allocated to the order line is given by the minimum of q and q'. If $q >= q'$ then q' is the minimum and the stock requested is allocated else if $q < q'$ then the stock allocated is the stock available.

The Coloured Petri net with inscriptions is shown in figure 6.4.

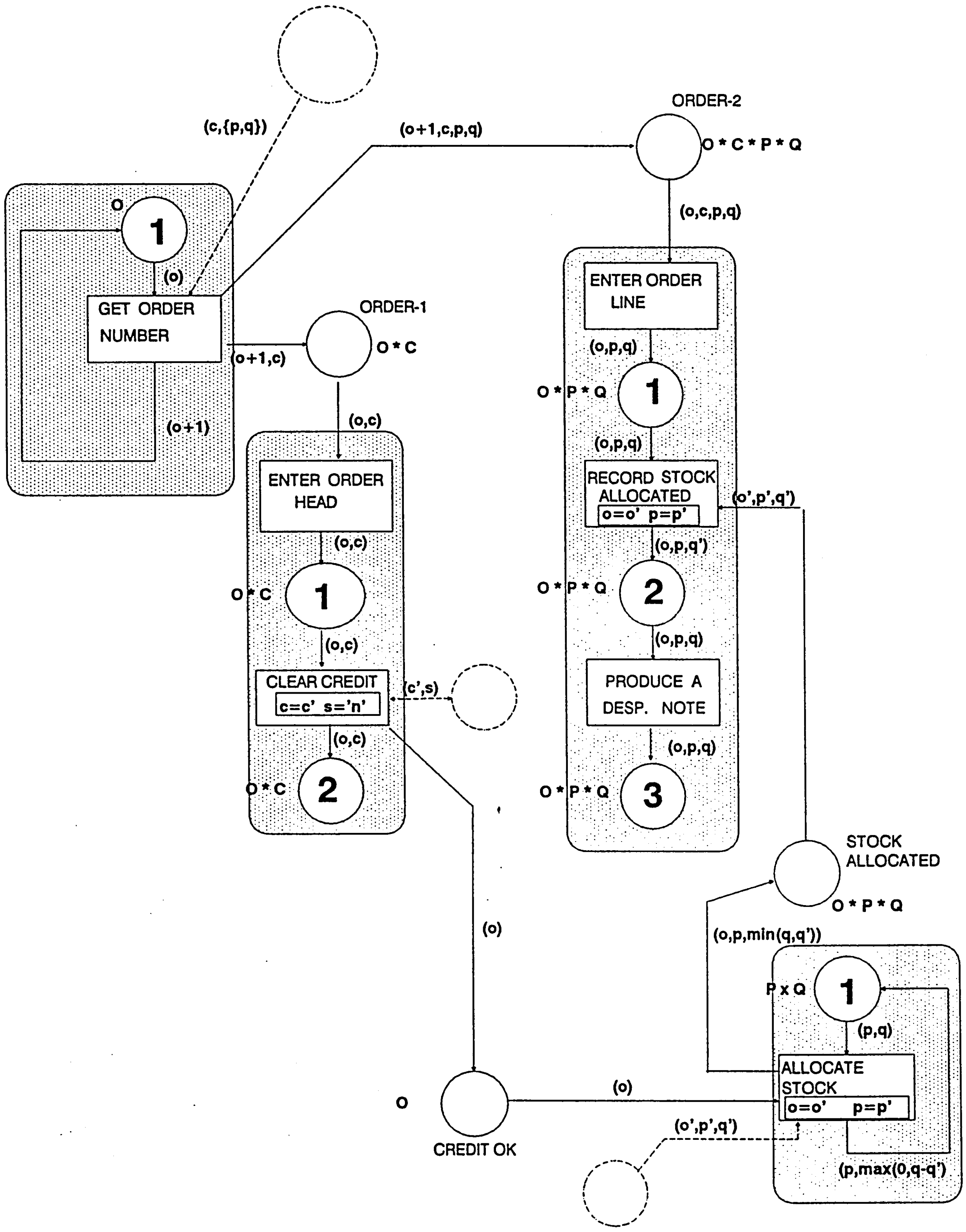

(c,{p,q})
(o+1,c,p,q)
ORDER-2
O * C * P * Q
(o,c,p,q)
O
1
(o)
GET ORDER NUMBER
ORDER-1
O * C
(o+1,c)
(o+1)
(o,c)
ENTER ORDER LINE
(o,p,q)
O * P * Q
1
(o,p,q)
RECORD STOCK ALLOCATED
o=o' p=p'
(o',p',q')
ENTER ORDER HEAD
(o,c)
O * C
1
(o,c)
CLEAR CREDIT
c=c' s='n'
(c',s)
(o,c)
O * C
2
(o,p,q')
O * P * Q
2
(o,p,q)
PRODUCE A DESP. NOTE
(o,p,q)
O * P * Q
3
STOCK ALLOCATED
O * P * Q
(o)
(o,p,min(q,q'))
P x Q
1
(p,q)
ALLOCATE STOCK
o=o' p=p'
O
(o)
(o',p',q')
CREDIT OK
(p,max(0,q-q')

Figure 6.4

6.2 Advantages of representing logical designs by Coloured Petri Nets

The major advantage is that the design embraces the notion of parallelism and therefore leads to implementation of information systems on parallel architectures. The creating of large complex logical designs is feasible. The analysis of such designs using the theory of Petri nets before implementation is vital with large, complex, parallel applications. Modelling with coloured Petri nets makes analysis possible.

6.3 System nets

6.3.1 Introduction

Simulation of coloured Petri net models require a mapping of each of the net elements into a real resource. To enable simulation system nets have been introduced which model programs and relational tables. Mapping rules have been developed from coloured Petri nets into system nets which are then implemented using the simulator. System nets [Cutts/Magee-89] are the notation for information system designs. They are constructed by applying transformation rules to Coloured Petri nets. System nets enable the construction of design schema embracing the notion of parallelism.

Figure 6.5 shows a System net.

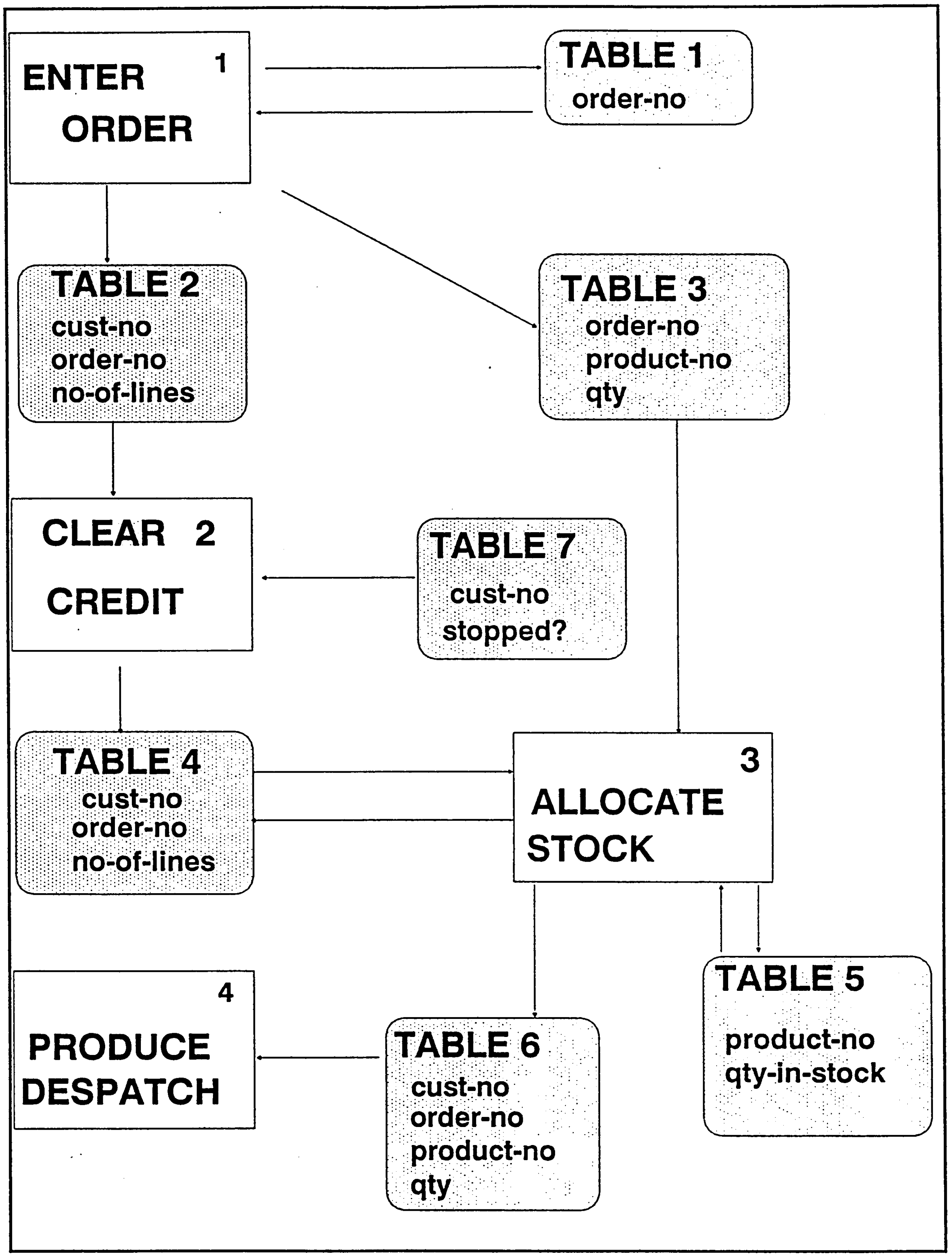

ENTER ORDER 1
TABLE 1 order-no
TABLE 2 cust-no order-no no-of-lines
TABLE 3 order-no product-no qty
CLEAR CREDIT 2
TABLE 7 cust-no stopped?
TABLE 4 cust-no order-no no-of-lines
3 ALLOCATE STOCK
4 PRODUCE DESPATCH
TABLE 6 cust-no order-no product-no qty
TABLE 5 product-no qty-in-stock

Figure 6.5

6.3.2 Definition of System nets

-places, representing relations and message buffers, drawn as shaded boxes,

-transitions, representing by executable processes,p, and executable tests,pt, drawn as boxes, and

-arrows, representing record input / output.

System nets use places to represent relations and message buffers. The colour functions represented on the coloured net are translated into relational tables. Each place on the system net is associated with a filename which contains details of the relational table's domains. System nets use transitions to represent events. Each transition is associated with two files. The first file contains a specification of the transition's enabling criteria. The enabling criteria embrace the guards and the availability of input (pre- conditions). The second file contains a specification of the event's action including the output (post-condition) and the expression functions. The first file is called the 'test' file and the second the 'process' file. System nets are a way of representing coloured nets in an implementable format..

6.4 Conclusions

Coloured Petri nets can be easily constructed from entity life histories and entity event diagrams. The use of coloured Petri nets to represent parallel information systems in graphical form has further proved to be beneficial as individual processing elements, tables and messages can be easily identified and studied. The translation to system nets proved to be a useful route through to simulation of the complete information system.

Chapter 7 INTEGRATED TOOLS

7.1 Introduction

An integrated support environment is a combination of software utilities that permit a system to be used to best effect for a particular purpose, in this case the development and maintenance of software based on a System net schema. The tool should enable specification and design documents to be created and edited quickly and easily in a graphical format, a textual format and a mathematical format. Moreover the tool should be able to transform one format to another. The tools must be capable of representing the characteristics of the application, principally structure, behaviour, communication and configuration.

Animation of a design is the process of providing an indication of the dynamic behaviour by walking through a specification fragment in order to follow some scenario. It involves stepping through each process action, examining the resultant output behaviour and interacting with the scenario when desired. Where appropriate, tools were selected from those available, however, it was necessary to develop tools for the analysis of entity life histories and for the simulation of system nets.

7.2 Automate Plus

Automate plus from LBMS was selected for development of data flow diagrams, entity event diagrams, entity models and their associated documentation. The author was familiar with Automate having used it extensively in consultancy and teaching work. There were no problems with its use for the production of diagrams and associated documentation.

Diagrams are stores within Automate as text files. This has one major disadvantage in that the diagrams must be reconstructed by Automate utilities every time they are displayed or printed. It was found that the text files have a standard format which can be analysed making it possible to write software to analyse entity event diagrams to identify the events, the entities, the input and output, data flows and the guards. This software written by the author in Pascal saved a considerable amount of time inspecting entity event diagrams for the significant items. The output also formed a major input to the transformation rules described in chapter 6. The selection and use of Automate Plus was very successful.

7.3 The Entity Life History Tool

The entity life history tool comprises two main functions, an editor and an analyser.

7.3.1 The Editor

The editor provides three major functions.

1. Loading, saving and renaming ELH files.

2. ELH drawing and subsequent editing.

3. Plotting to paper size A3 or A4.

Functions 1 and 3 are straight forward. Edit provides six major functions connected to the three icons, circle, rectangle and arc. They are, add a place, add a transition, add an arc, scaling, moving icons around the screen and editing labels and text.

Figure 7.1 shows an ELH entered into the screen.

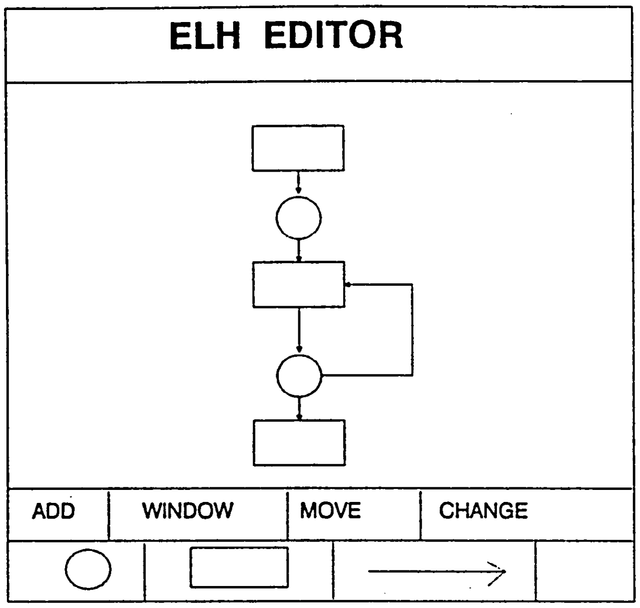


Figure 7.1

The ELH editor provides a simple interface for the graphical construction and amendment of entity life histories. The editor was built to provide a similar user interface to Automate Plus. It is a mouse driven editor with facilities to select and position icons on the screen.

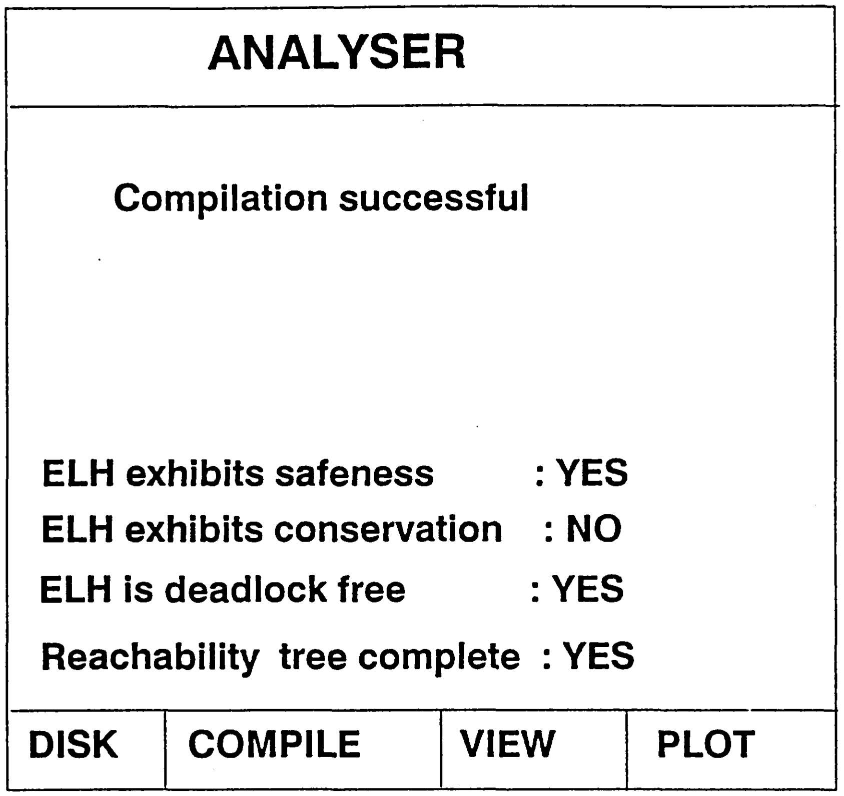
The moving and scaling options proved to be satisfactory. The ability to scale by decimal factors prooved particularly useful, a facility which was not available in Automate. Generally the software proved to be reliable for small entity life histories, it did however cause major difficulties with large entity life histories and could not be recommended for use with large files. Significantly commercial software is now available for the construction of place transition nets which could be adapted easily for development of entity life histories. The major problem with the software was its ability to handle large files. The software also proved to be frustrating to use. The graphics produced by Pascal were no where up to Automate standards and the speed of response to the mouse was slow. An understanding of the problems in this area led to a rewrite of the software in C which overcame these problems.

The specification of the tool envisaged a general purpose tool. The consequence of this was that many of the parameters had to specified over and over again when their use for entity life histories was standard. Examples of this are place capacities and arc weights which for entity life histories are always 1. The tool as specified prompted the user to input capacities and weights, which, in the entity life history case, was always 1. This again proved to be frustrating.

7.3.2. The Analyser

The analyser provides identical functions for loading and saving ELH. Its major function is via a COMPILE operation which provides the analysis facilities. COMPILE takes a loaded ELH and analyses it for deadlock, safeness, conservation and boundedness. COMPILE also creates the reachability tree which can be displayed on the screen using the VIEW option. The reachability tree may be saved alongside the ELH.

Figure 7.2 shows an example COMPILE screen and figure 7.3 an example reachability tree.

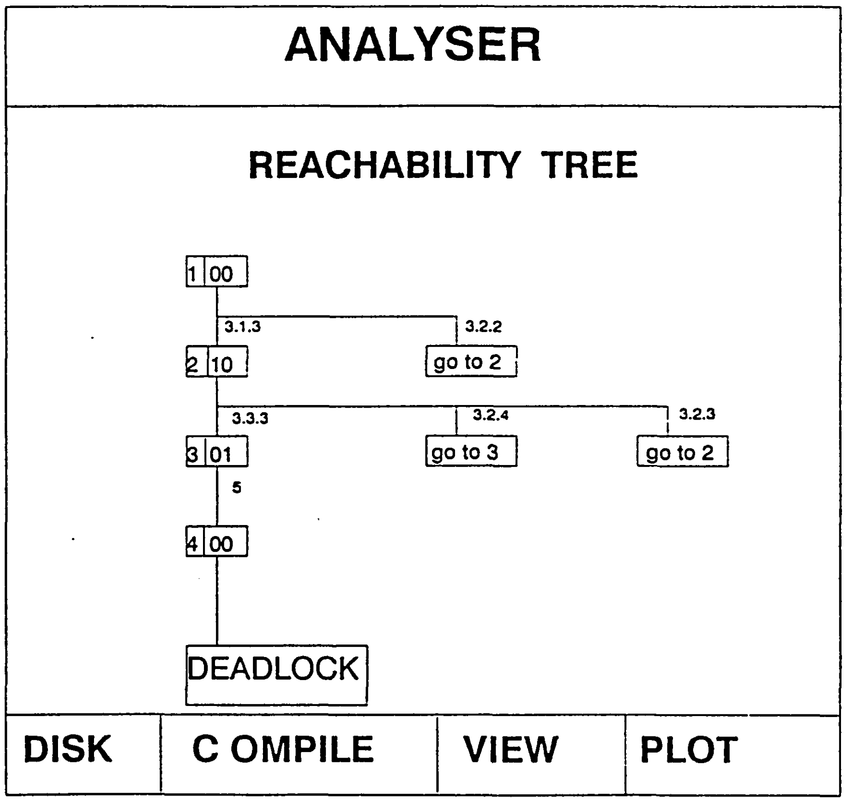


ANALYSER

Compilation successful

ELH exhibits safeness : YES
ELH exhibits conservation : NO
ELH is deadlock free : YES
Reachability tree complete : YES

Figure 7.2



ANALYSER

REACHABILITY TREE

Figure 7.3

The analyser provides an easy way to determine if ELH are correct. There were no problems with the analyser and very valuable results were obtained from its use. This software was again written in Pascal, it was slow but had no other drawbacks. The major problem was that it depended on an entity life history being input via the editor.

7.4 The Application Development Environment

7.4.1 Introduction

A total environment was created to support the development of parallel information systems. The environment implements a System net design and provides a simulation tool comprising:-

1. A System net editor.
2. A database create, modify and view facility.
3. A program development facility.
4. A program test environment.
5. A system test environment.
6. An application simulation environment in which the real information system executes on a single processor.

Places are implemented using a relational database, transitions are implemented by separate executable files. Each transition comprises four files, its test source code file, its test executable file, its process source code file and its process executable file. Output flows, i.e. flows from processes to tables are implemented within the process. The test program, for each process, determines if input is available and if its guard is true and therefore whether the process should execute. This is equivalent to binding. The 'test' code evaluates the binding and returns a true or false flag. If the flag is true then the process code is enabled.

7.4.2 The System Net Editor

The net editor allows System nets to be easily constructed using graphics and positioning of icons by mouse. It provides for the addition of processes (transitions), tables (places), links (arcs) and text, deletion of a process or a table, window adjustment, icon size adjustment, retrieval of a net from disk and writing of a net to disk.

These facilities allow a net structure to be input, modified and saved. The lessons learned from construction of the entity life history editor were used to overcome the problems experienced. The software was written in C with standard graphics routines, sophisticated scaling and the ability to construct a net over many screens using the physical screen as a window. The tool is quick, it uses mouse driven menus and is very easy to use. System nets were input and edited very easily using the tool. Additional information may be added to the net using edit functions. These functions link the structure of the net to program and data files to enable the simulator to execute programs for transition occurrence and to pick up data from places represented as tables.

Each place is instantiated as a relational table and given a filename. This filename is associated with the place and stored as part of the net. Further utilities allow tables to be created and modified, and table contents to be examined.

Name, edit and compile options exist for both the process and test functions associated with a transition. The first option associates a filename with the source code and executable code. The file names are stored with the net for each process. Each process, therefore, has two source code files and two executable files. The second option allows the source code files to be edited. Source code programs may now be created or modified using the editor's facilities. The third option allows compilation of the source code. The executable code is written to the executable code filename. These names were associated with the process on the net.

The edit process function therefore associates with process, two source code files and two executable code files. This code may be in any language as all code files are independent. These facilities allow each of the System net structures to be associated with resources to be used during simulation.

There were no problems with these processes, the software proved to be particularly easy to use. The ability to link directly from the System net structure on screen directly to table create and modify, to program editors and compilers, to an execution facility and then to be able to examine the table contents provided an excellent interactive, incremental program test facility, small scale prototyper and a means by which large systems could be developed incrementally. Much of the success of this tool was due to the developers interface which linked all these facilities.

The development program comprises a three window screen, the graphics area where System nets are constructed and displayed, a menu area with selection by mouse and an input area for text, eg. filenames. The system is clear and easy to use.

7.4.3 The Test and Simulation Environment

This is the crux of the system and perhaps the total research programme. The software provided a test and simulation environment for parallel information systems. All of the facilities are available from an Execute menu. One facility, GO performs a number of functions.

1. The System net is displayed on the screen. At this stage SELECT & EXAMINE may be used to determine the exact state of all the tables.

2. The TEST CYCLE executes all test.exe files, ie. all tests are performed. According to their return, true or false a table of enabled processes is displayed.

```
TEST CYCLE

Testing  P1 enabled

Testing  P2 not enabled

Testing  P3 not enabled

Testing  P4 not enabled

Testing complete - press left mouse to proceed
```

Pressing the right mouse at this point returns the execute to the menu screen. It may be processes enabled/not enabled are not as expected and the SELECT/EXAMINE process may need to be repeated. It may be necessary to return & examine the test programs, this is possible, indeed programs could be edited and recompiled before testing continues. Pressing the left mouse continues the execute phase.

3. The Process Cycle displays the System net on the screen with the mouse pointer pointing to the first enabled process.

Left mouse invokes the process's executable file (right mouse returns to menu), which executes and may adjust table contents. Any executable code in a separate named file is acceptable, the simulator will work with test and process code in any language. On completion of execution the cycle pauses, 'execution of process n complete.' Right mouse now returns to the menu so that tables may be examined to ascertain that the process has been executed correctly; left mouse invokes the next enabled process. When all enabled processes have been executed the system returns to the test cycle.

This method of examining the net structure for enabled processes, followed by inspection then execution of enabled programs one at a time with possible inspection of tables between each execution made testing and modification significantly easier than any other environment I have ever encountered. The production of tested code and tables was very rapid, no problems were experienced and because the structure of the information system was captured by the system net the information system grew incrementally using small scale prototypes. Because the underlying structure was correct then the system was correct as it was constructed and integration testing was not required. This achieved a major objective, that of 'correct by construction'.

The test and execute cycles contain several pauses where right mouse will return the system to the menu. If 'no pause' is selected the system will continuously alternate test and execute cycles until no processes are enabled. This option runs the information system comprising relational tables and individual code files which manipulate tables. All code files are independent and only pass data and control via tables. This is a very important feature when considering parallel execution since all enabled processes are enabled in parallel ie. they may execute concurrently. On a single processor each process is executed in turn, on a parallel architecture machine with multiple processors all processes could execute in parallel. This is the vital point of the design.

This final simulation of the information system on a single processor proves the correctness of the design. In fact the information system works as required and very considerable benefit accrues from the approach even when implementation is not on a parallel architecture. The ease of development makes this approach an important contributor to the development of large complex information systems.

7.5 Conclusions

This design environment and tool demonstrates that the construction and implementation of System nets is very easy. An information system comprising over fifty tables and thirty programs has been implemented successfully. This application is described in detail in chapter 8.

Chapter 8 CASE WORK

8.1 Introduction

The case work is based on a case study supplied by Learmonth and Burchett Management Systems (LBMS). The case is the Duet Wright Consulting Company. Section 8.2 provides the case study scenario. The data modelling aspects of the case study were relatively straight forward. The case work included the entity event diagrams, the entity event matrix, the entity life histories, the coloured Petri net, the System net and a part implementation.

8.2 The Duet Wright Case [LBMS]

8.2.1 Introduction

In 1980, two brothers named Wright (not Orville and Wilbur) who were both in the data processing business, decided to join forces in the spirit of true entrepreneurship, and started a consulting company. They named the company Duet Wright, not only because there were two of them and their last name was Wright, but also because they felt the name reflected what they hoped to accomplish with each project they took on. When the company consisted solely of the two brothers with a few projects, the manual system for doing business was sufficient. As business and staff grew, the manual system could no longer handle the load. They have now requested an automated system to track projects, consultants, clients and time reporting.

8.2.2 Description of the Business Environment

The company has defined a number of standard skills, i.e. IDMS, ADABAS, Logical Design, Teaching, etc. Each skill has a unique skill code. Any consultant could have a range of skills and a skill may be held by many consultants.

The company has a scale of consultant grades ranging from Managing Consultant as the top grade to Associate Consultant at the bottom. Each consultant belongs to only one grade at any time. Grades are identified by unique grade codes. Each consultant is allocated a unique personnel number. The company has four divisions and each consultant belongs to only one division at a time. Each division has a unique division number and can have many consultants.

Each client is given a unique client reference. Each project undertaken is given a project reference unique within the client reference. A project is related to only one client, but a client could be related to many projects. There will be a requirement to find all clients who are active (have outstanding projects) e.g. only inactive clients may be cancelled.

At any time, a consultant could be assigned to work on many projects and a project could have more than one consultant. When defining a project, the consulting manager will want to relate the requirement for specific skills to the project. A project may require many skills and any skill may be needed on many projects. Each week, consultants send in time sheets, identified by the consultant's personnel number and the week ending date. Time may be charged to many projects on a single time sheet and a project may take more than one week to complete. Consultants may only charge time to projects to which they have been assigned.

8.2.3 Systems Overview

The managing consultant will maintain a variety of information in the system, including:

- Consultant Data

- Details of Projects

- Client Information

The Consultant data will also be maintained by the Personnel Department to ensure accuracy.

When a project is to be started, Assignments will be entered on- line by the Managing Consultant for those Consultants who are available and have the appropriate skills required for the project. A monthly assignment report will be produced for review by the Managing Consultant and monthly schedule reports will be issued to the Consultants. Consultants charge time to the project using time sheets which are sent to accounting ever week. The time sheets will be input and edited on-line to ensure that the Consultant has been assigned to the project. Any erroneous time sheets will be sent back to the Consultant for correction. A Time sheet Summary may be produced monthly for Accounting to ensure all time sheets have been submitted. Every month, the time sheets are extracted and consolidated with rate information to produce a billing report which Accounting uses to produce client invoices.

8.2.4 Detailed Processing Requirements

The computer system defines the systems boundary.

Consultant, Project, and Client data are independently maintained on the system.

Consultant and Project data will be used to assign Consultants to projects.

Consultant data will include billing rates to produce the billing data for Accounting.

The personnel Department may update (add, modify or delete) all consultant data, Managing Consultants are only permitted to modify the Consultant records.

> Additions - when a Consultant is added to the system, the Personnel Department will enter the basic details
>
> Modification - include changes to basic details (e.g., name and address) as

well as the entry of a skill profile and changes in grade and division. If necessary, new standard skills may be identified.

Deletions - this flags the Consultants who have left the company as cancelled, so that they will be removed from the system by the monthly housekeeping run.

The Consultant add, change and delete transactions are kept in a transaction file and later used to produce the batch action summary.

When a project is proposed, the Managing Consultant will enter the project details into the system, ensure that the Client exists, and if necessary, flag the client as 'active'. The project profile may subsequently be extended through the use of 'modify project' function to include details of expected hours and skills required, etc. If a project is cancelled, it will be flagged as such on the system and the client status updated (to inactive) if appropriate.

The Accounting Clerk receives time sheets from Consultants on a weekly basis. These are entered into the system and validated against consultant, project, and assignment details. The system will maintain all time sheets on the time sheet file. Only when a time sheet has been input correctly will the project and assignment records be updated to reflect time spent to date. On a monthly basis, a time sheet summary may be produced for the Accounting Department and the Managing Consultants. This information includes Consultant and time sheet details and is used by the managers to indicate to the system that the time sheets are approved and ready for client billing.

8.3 The Logical Data Model

Figure 8.1 shows the logical data model.

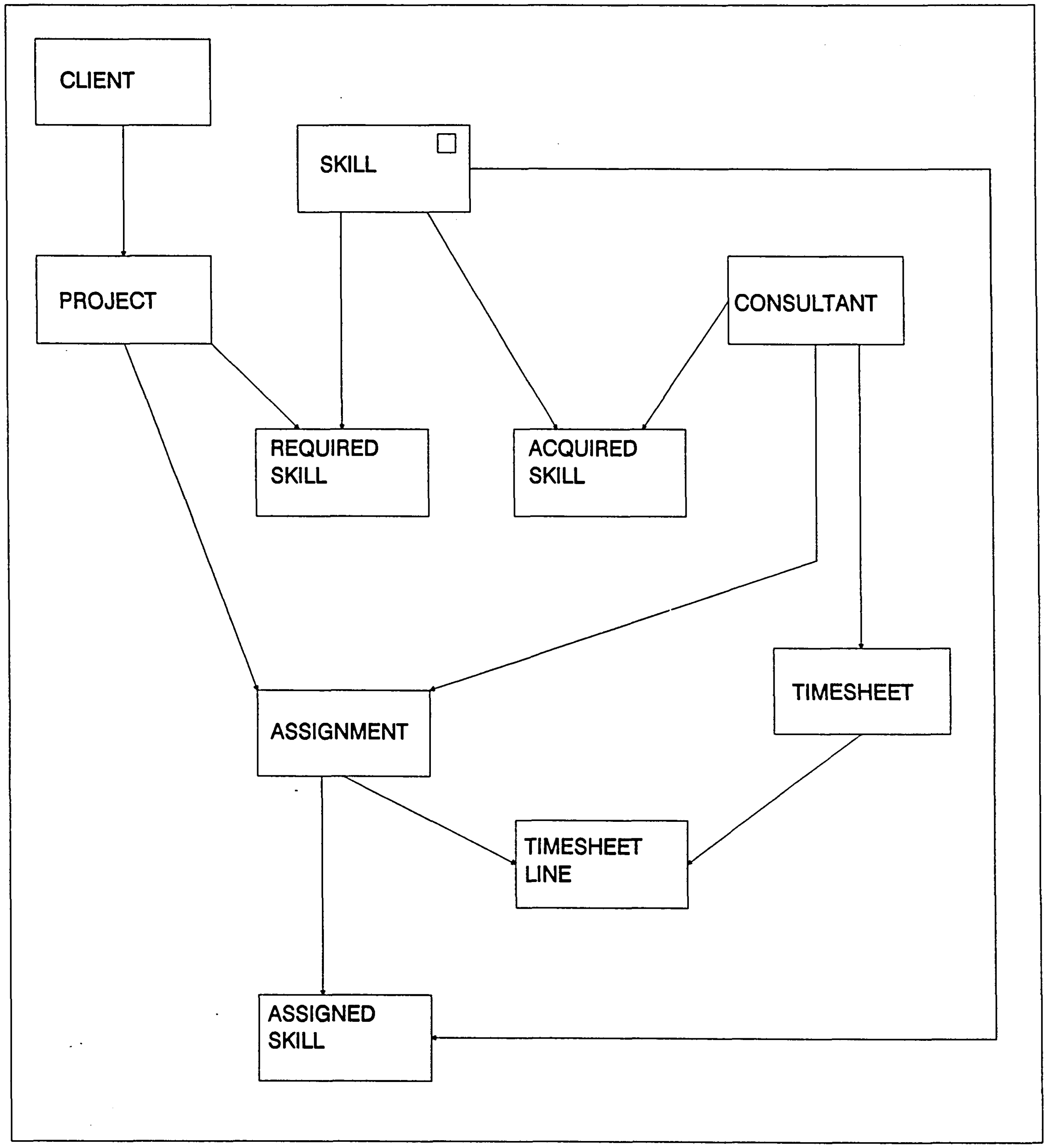


Figure 8.1

The data items are shown in figure 8.2.

PROJECT

Client#, project#, name, start date, expected hours, hours to date

REQUIRED SKILL

client#, project#, skill#, hours required, required date, comments

ASSIGNMENT

client#, project#, person#, expected start date, expected hours, role

ASSIGNED SKILL

client#, project#, skill#, person#

CONSULTANT

person#, name, grade, division, billing rate, address

ACQUIRED SKILL

skill#, person#, date acquired

CLIENT

client#, name, address, contact, status, 'phone#

TIME SHEET

person#, date weekend

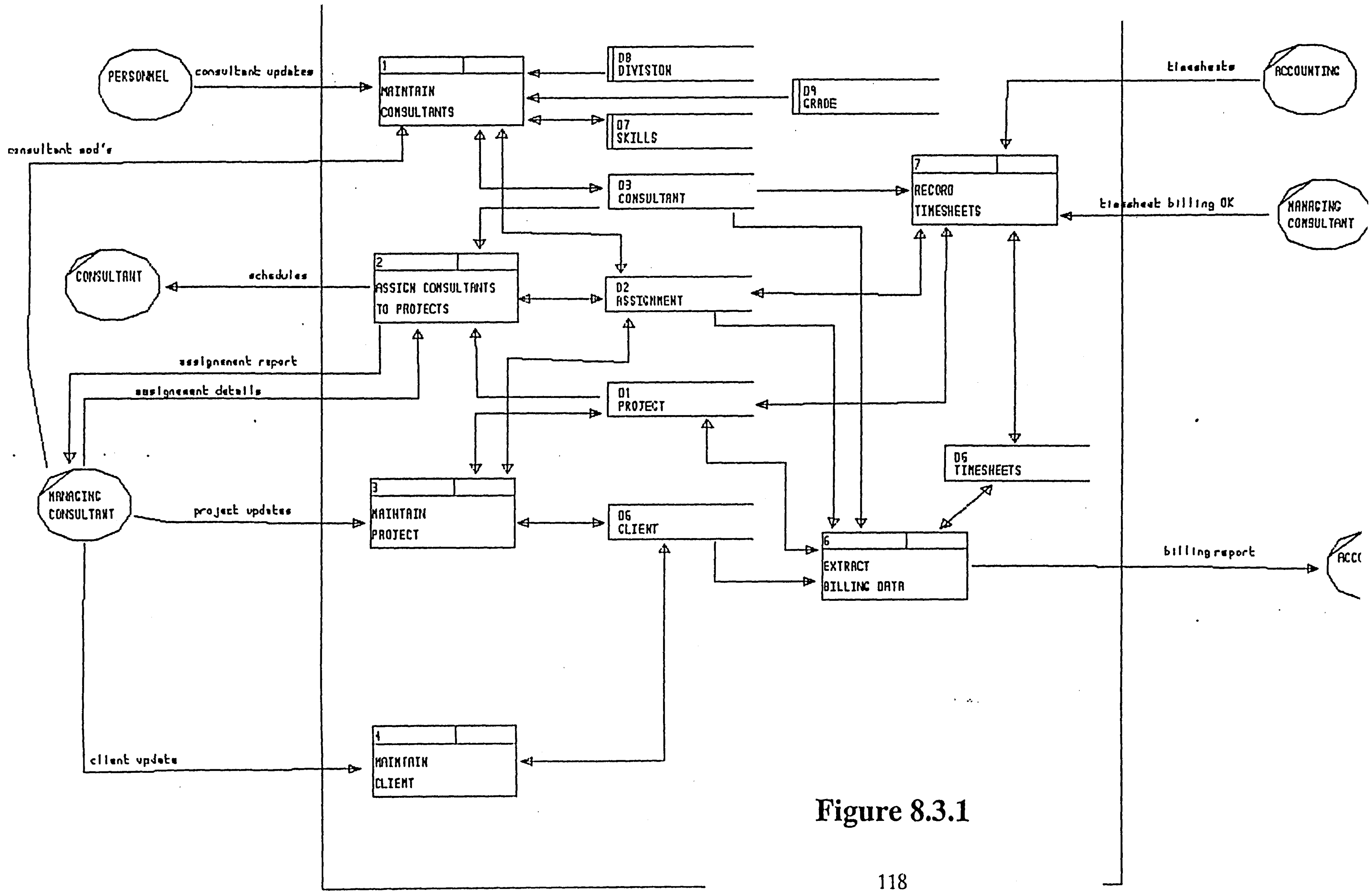
TIME SHEET LINE

person#, date, client#, project#, hours this week, hours to go, expected completion date

Figure 8.2

8.4 The Entity Event Diagrams

The entity event diagrams are shown in figure 8.3. There are 18 diagrams.



PERSONNEL
consultant updates
1 MAINTAIN CONSULTANTS
D8 DIVISION
D9 GRADE
D7 SKILLS
timesheets
ACCOUNTING
consultant mod's
D3 CONSULTANT
7 RECORD TIMESHEETS
timesheet billing OK
MANAGING CONSULTANT
CONSULTANT
schedules
2 ASSIGN CONSULTANTS TO PROJECTS
D2 ASSIGNMENT
assignment report
assignment details
D1 PROJECT
D5 TIMESHEETS
MANAGING CONSULTANT
project updates
3 MAINTAIN PROJECT
D6 CLIENT
6 EXTRACT BILLING DATA
billing report
ACC
client update
4 MAINTAIN CLIENT

Figure 8.3.1

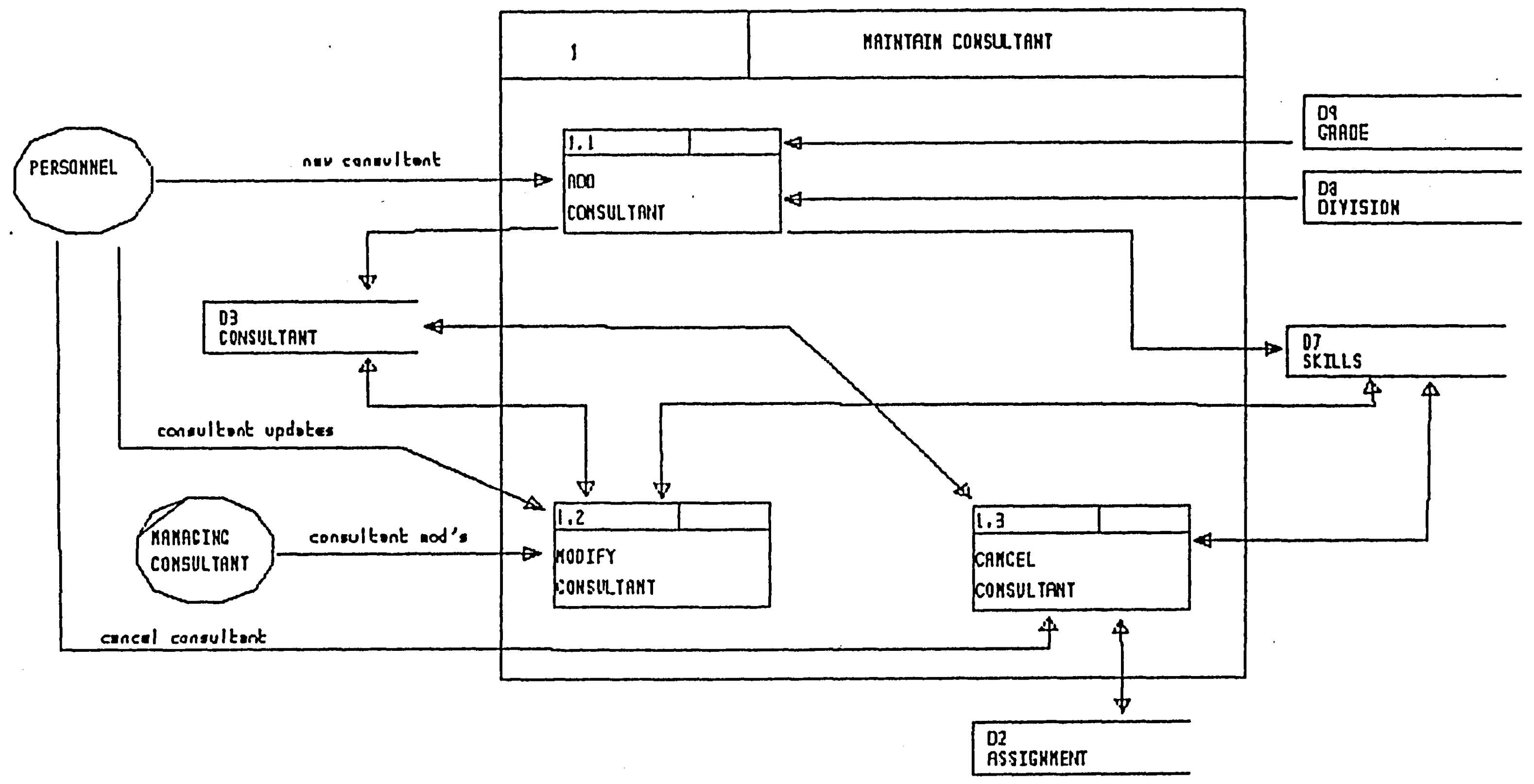

1
MAINTAIN CONSULTANT
PERSONNEL
new consultant
1.1
ADD
CONSULTANT
D9
GRADE
D8
DIVISION
D3
CONSULTANT
D7
SKILLS
consultant updates
MANAGING
CONSULTANT
consultant mod's
1.2
MODIFY
CONSULTANT
1.3
CANCEL
CONSULTANT
cancel consultant
D2
ASSIGNMENT

Figure 8.3.2

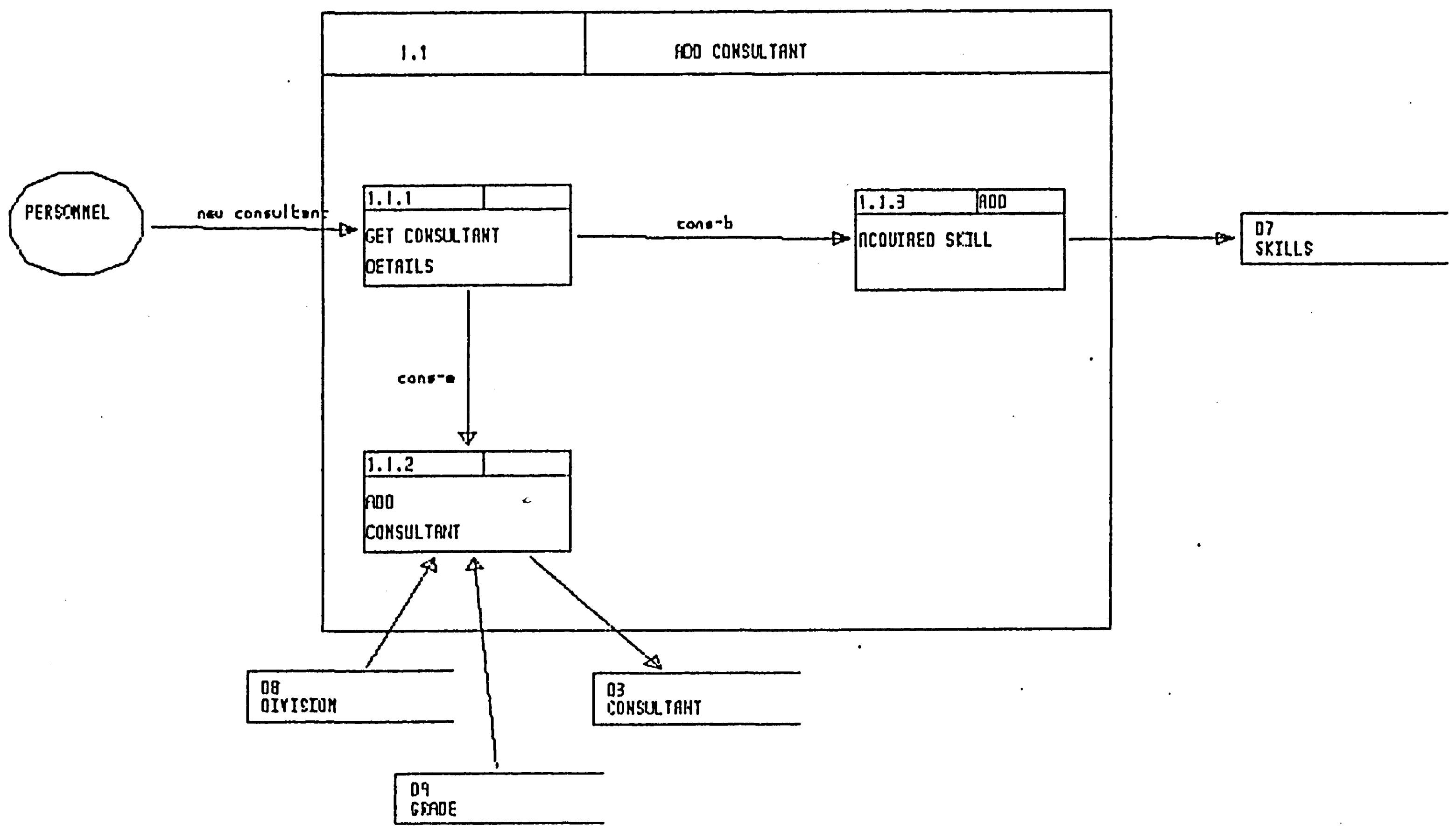

PERSONNEL
new consultant
1.1
ADD CONSULTANT
1.1.1
GET CONSULTANT DETAILS
cons-b
1.1.3
ADD
ACQUIRED SKILL
D7 SKILLS
cons-a
1.1.2
ADD CONSULTANT
D8 DIVISION
D9 GRADE
D3 CONSULTANT

Figure 8.3.3

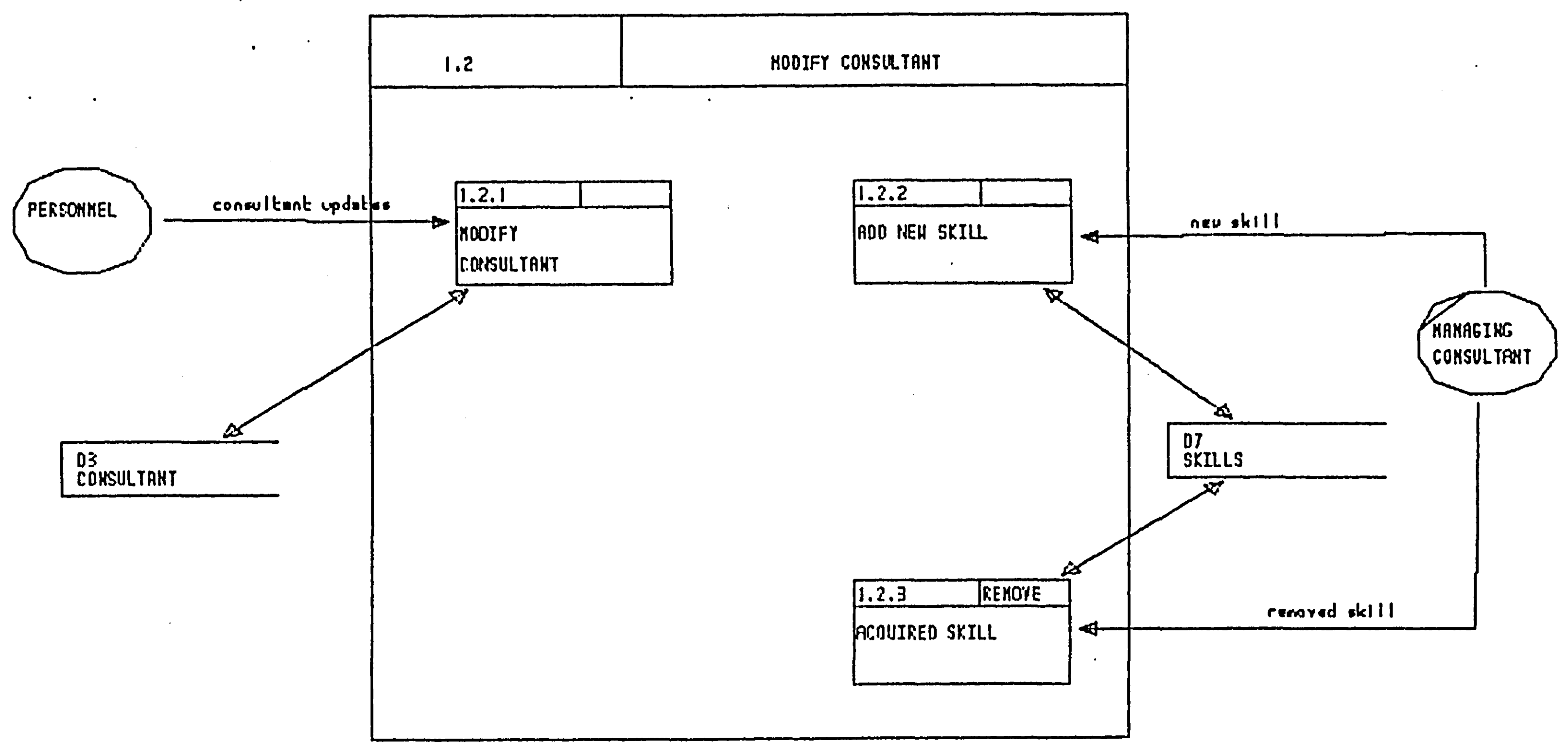

1.2
MODIFY CONSULTANT
PERSONNEL
consultant updates
1.2.1
MODIFY CONSULTANT
D3 CONSULTANT
1.2.2
ADD NEW SKILL
new skill
MANAGING CONSULTANT
D7 SKILLS
1.2.3
REMOVE
ACQUIRED SKILL
removed skill

Figure 8.3.4

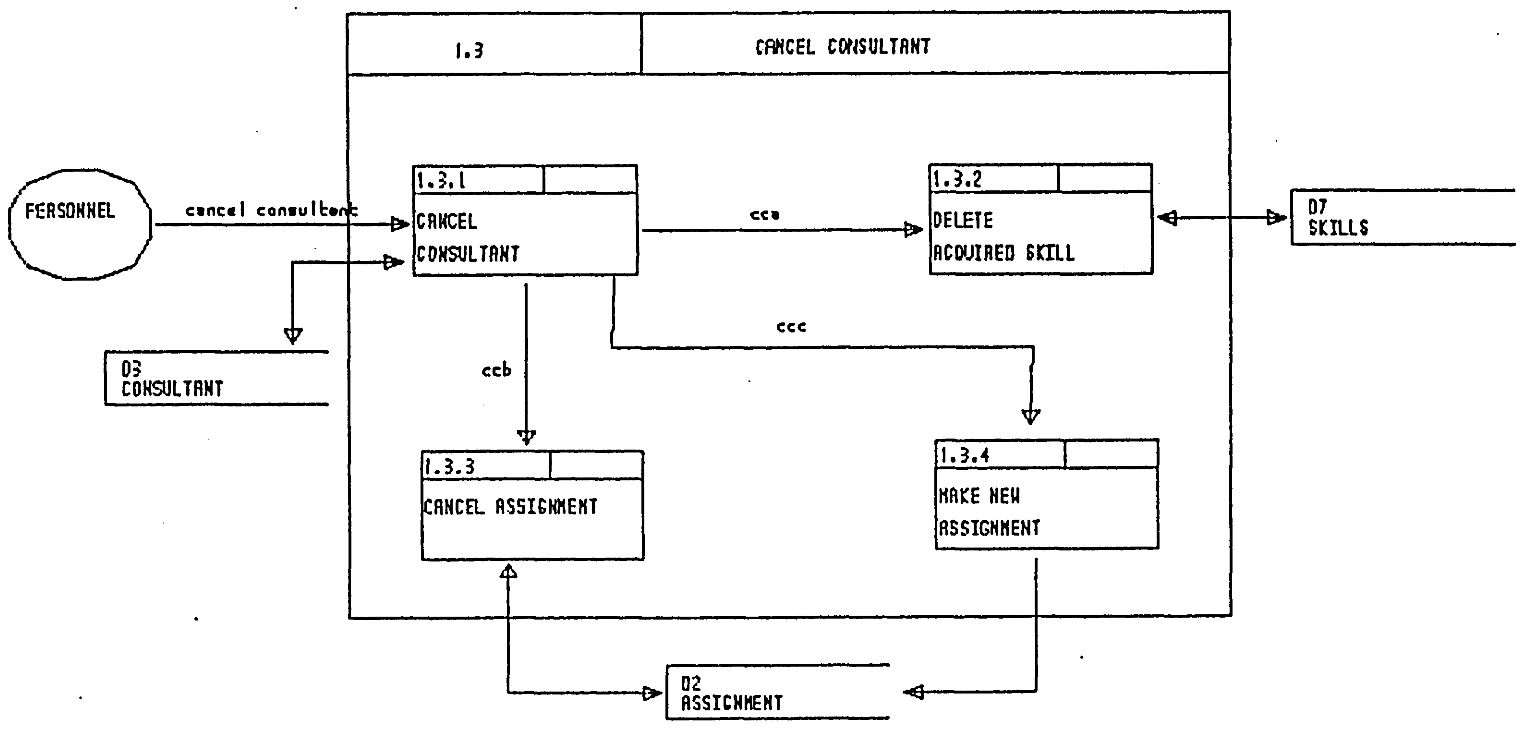

1.3
CANCEL CONSULTANT
PERSONNEL
cancel consultant
1.3.1
CANCEL
CONSULTANT
cca
1.3.2
DELETE
ACQUIRED SKILL
D7
SKILLS
D3
CONSULTANT
ccb
ccc
1.3.3
CANCEL ASSIGNMENT
1.3.4
MAKE NEW
ASSIGNMENT
D2
ASSIGNMENT

Figure 8.3.5

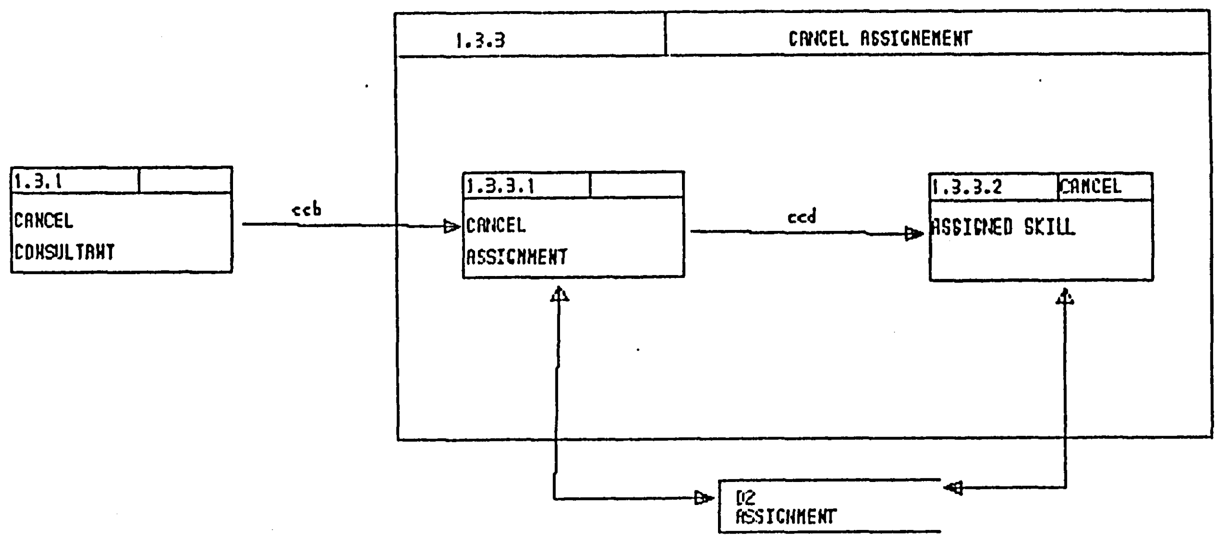

1.3.1
CANCEL
CONSULTANT
ccb
1.3.3
CANCEL ASSIGNEMENT
1.3.3.1
CANCEL
ASSIGNMENT
ccd
1.3.3.2
CANCEL
ASSIGNED SKILL
D2
ASSIGNMENT

Figure 8.3.6

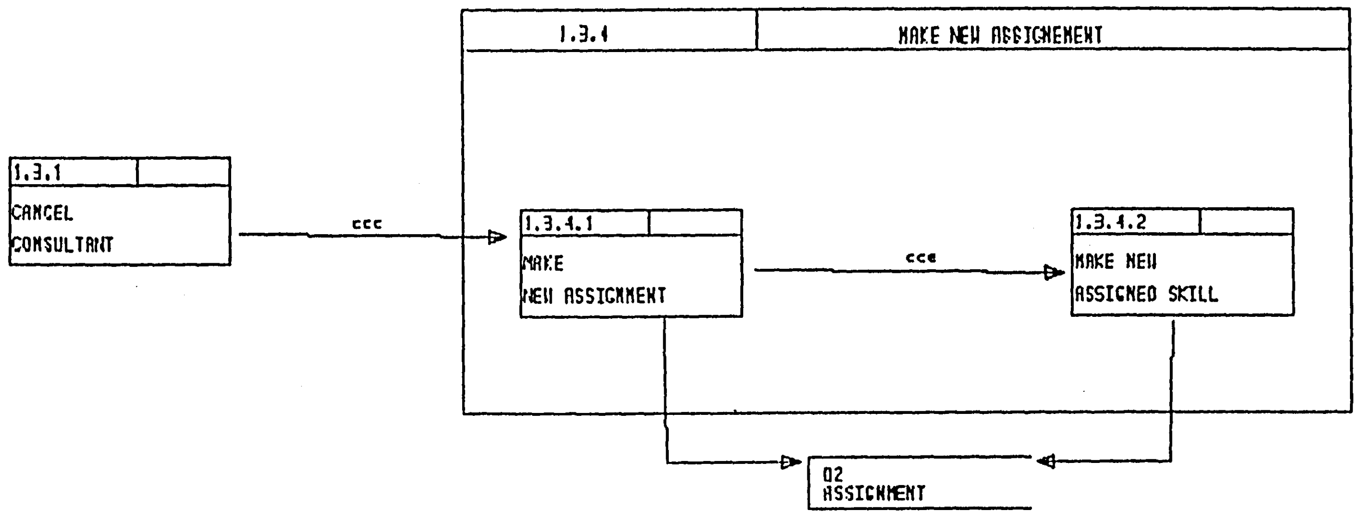

1.3.1
CANCEL
CONSULTANT
ccc
1.3.4
MAKE NEW ASSIGNEMENT
1.3.4.1
MAKE
NEW ASSIGNMENT
cce
1.3.4.2
MAKE NEW
ASSIGNED SKILL
D2
ASSIGNMENT

Figure 8.3.7

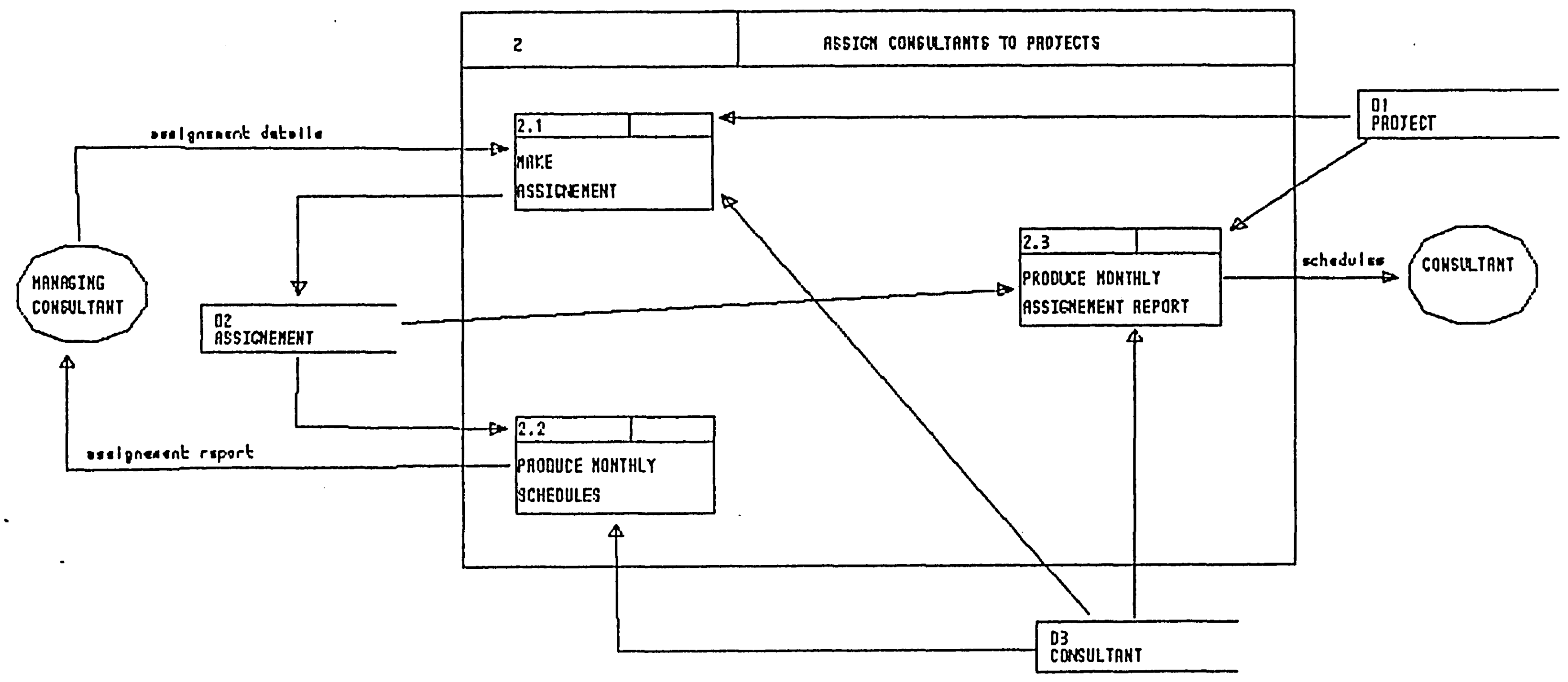

2
ASSIGN CONSULTANTS TO PROJECTS
2.1
MAKE ASSIGNEMENT
2.2
PRODUCE MONTHLY SCHEDULES
2.3
PRODUCE MONTHLY ASSIGNEMENT REPORT
D1 PROJECT
D2 ASSIGNEMENT
D3 CONSULTANT
MANAGING CONSULTANT
CONSULTANT
assignement details
assignement report
schedules

Figure 8.3.8

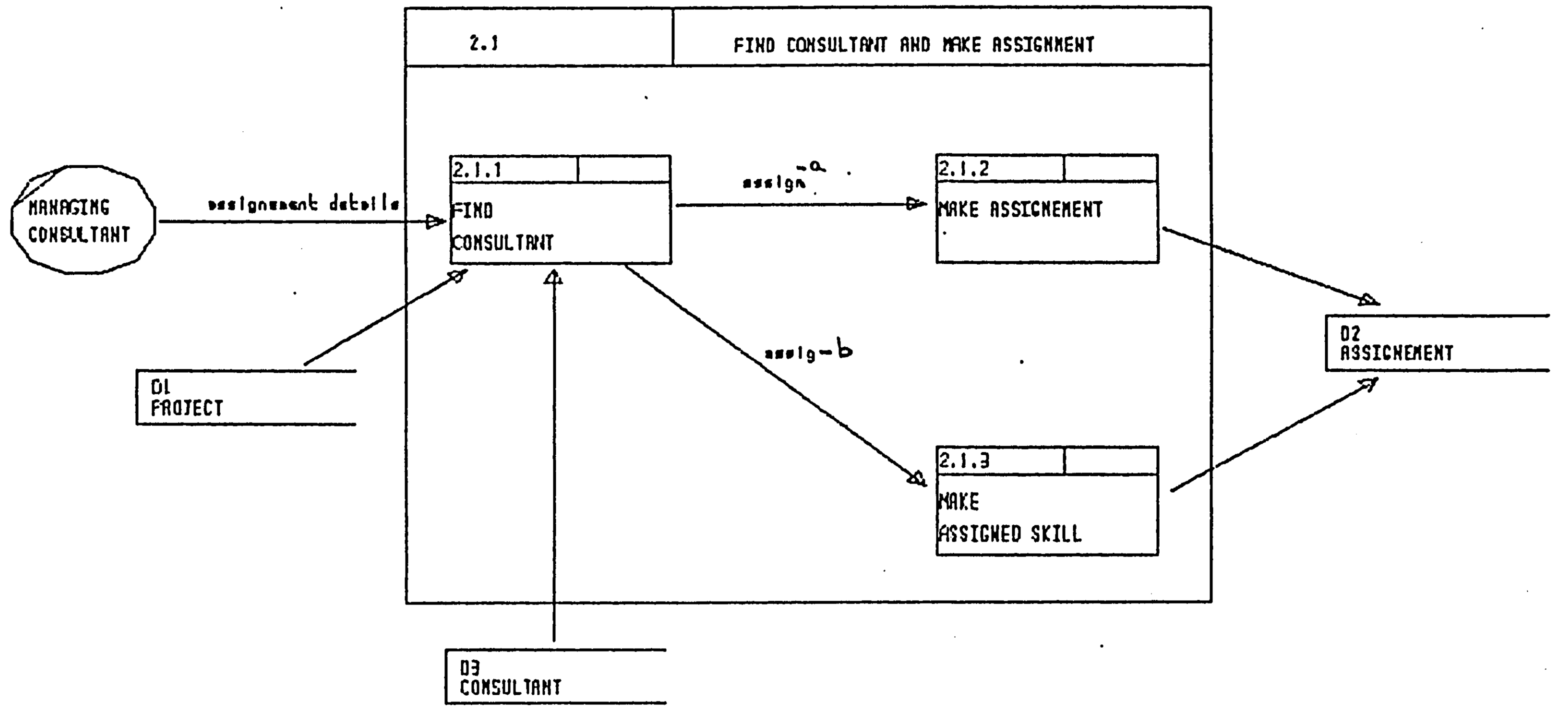

2.1
FIND CONSULTANT AND MAKE ASSIGNMENT
MANAGING CONSULTANT
assignment details
2.1.1
FIND CONSULTANT
assign-a
2.1.2
MAKE ASSIGNMENT
assign-b
2.1.3
MAKE ASSIGNED SKILL
D1 PROJECT
D2 ASSIGNMENT
D3 CONSULTANT

Figure 8.3.9

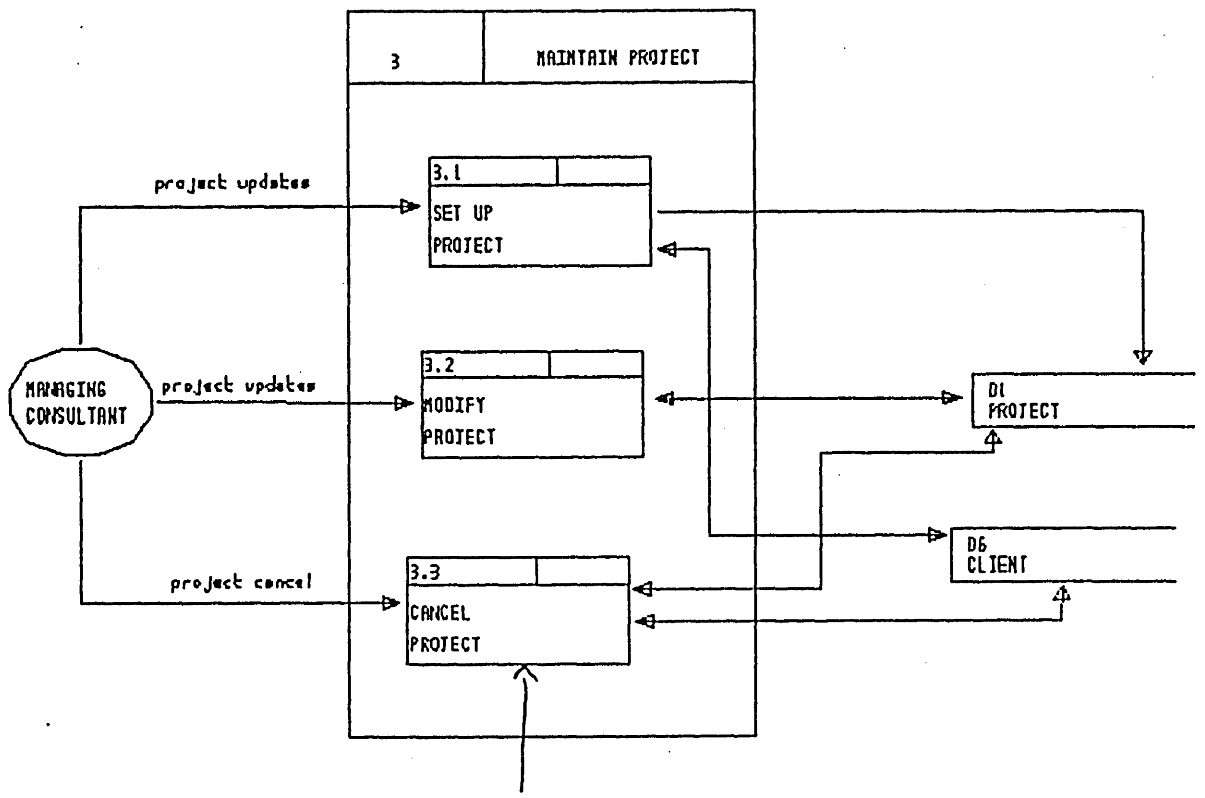

3
MAINTAIN PROJECT
project updates
3.1
SET UP
PROJECT
MANAGING
CONSULTANT
project updates
3.2
MODIFY
PROJECT
project cancel
3.3
CANCEL
PROJECT
D1
PROJECT
D6
CLIENT

Figure 8.3.10

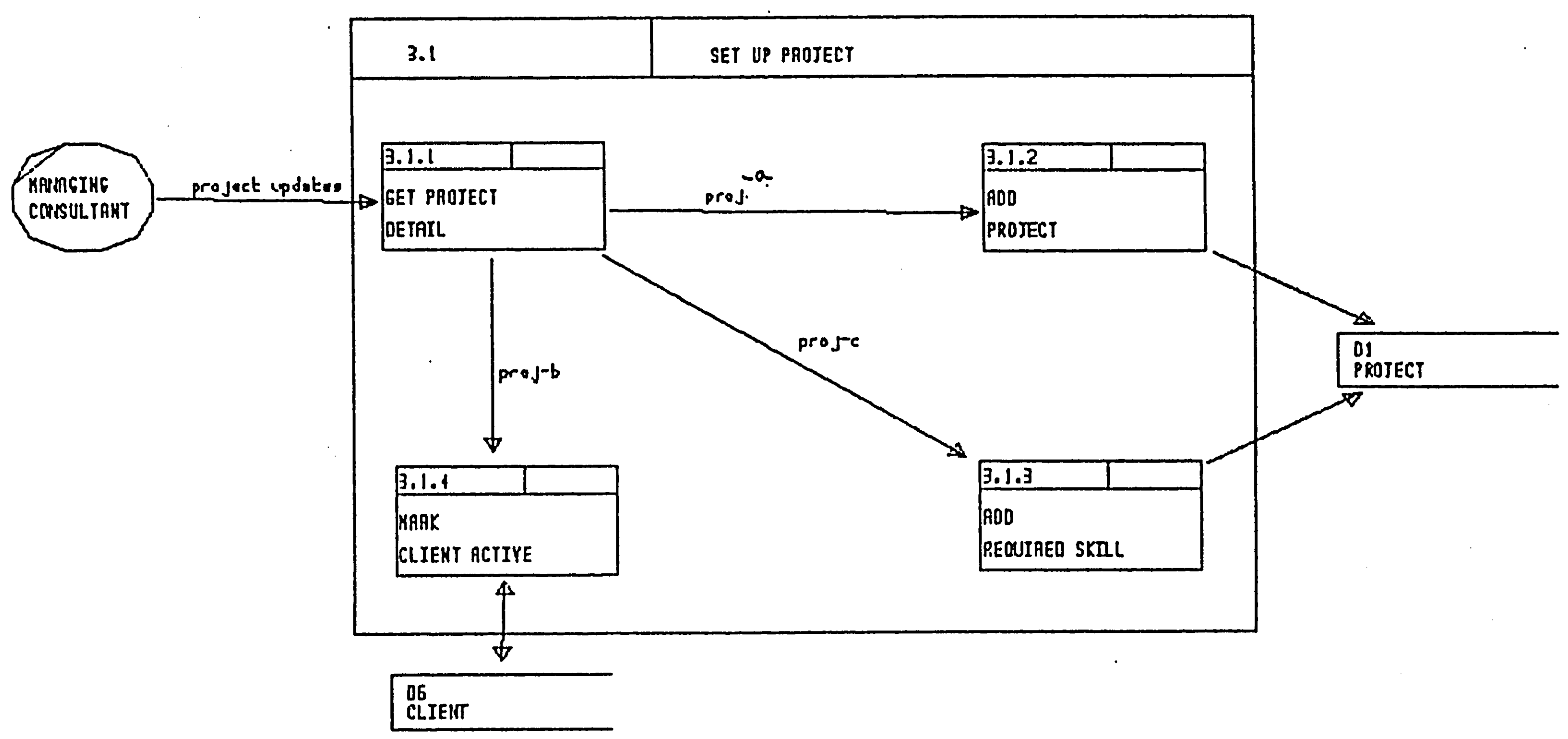

MANAGING CONSULTANT
project updates
3.1
SET UP PROJECT
3.1.1
GET PROJECT DETAIL
proj-a
3.1.2
ADD PROJECT
proj-b
proj-c
3.1.4
MARK CLIENT ACTIVE
3.1.3
ADD REQUIRED SKILL
D1 PROJECT
D6 CLIENT

Figure 8.3.11

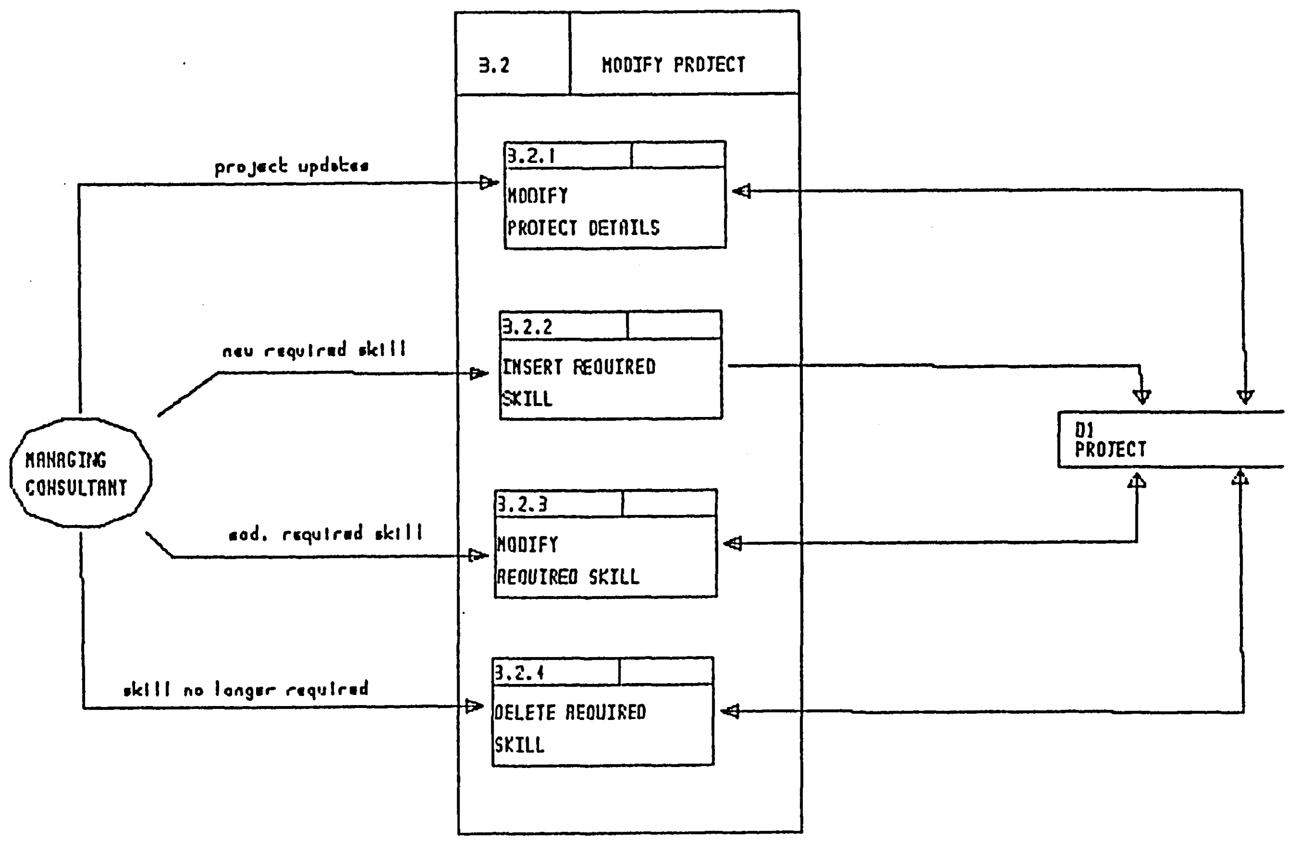

3.2
MODIFY PROJECT
project updates
3.2.1
MODIFY
PROJECT DETAILS
new required skill
3.2.2
INSERT REQUIRED
SKILL
MANAGING
CONSULTANT
D1
PROJECT
mod. required skill
3.2.3
MODIFY
REQUIRED SKILL
skill no longer required
3.2.4
DELETE REQUIRED
SKILL

Figure 8.3.12

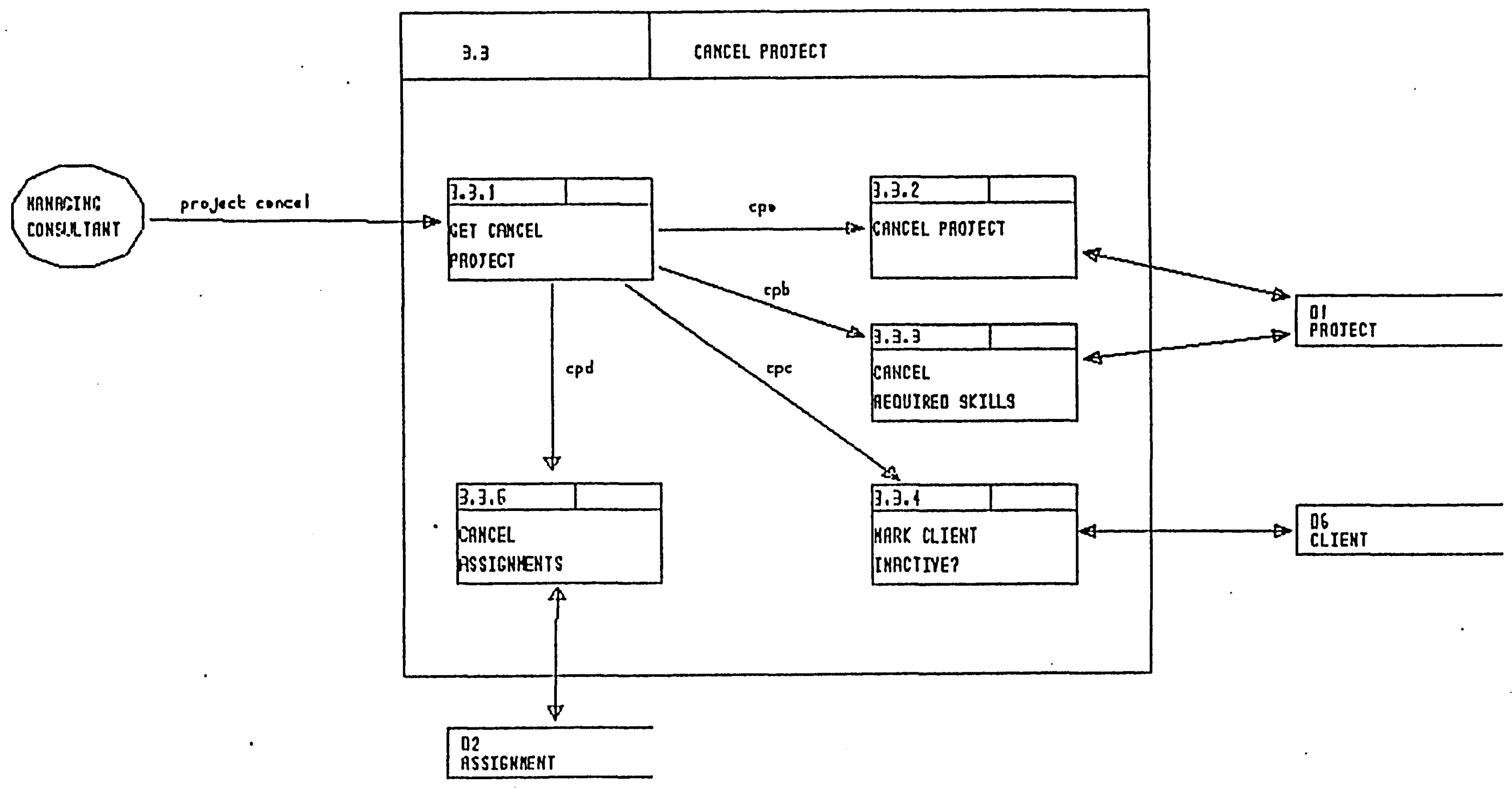

3.3
CANCEL PROJECT
MANAGING CONSULTANT
project cancel
3.3.1
GET CANCEL PROJECT
cpa
cpb
cpc
cpd
3.3.2
CANCEL PROJECT
3.3.3
CANCEL REQUIRED SKILLS
3.3.4
MARK CLIENT INACTIVE?
3.3.6
CANCEL ASSIGNMENTS
D1
PROJECT
D6
CLIENT
D2
ASSIGNMENT

Figure 8.3.13

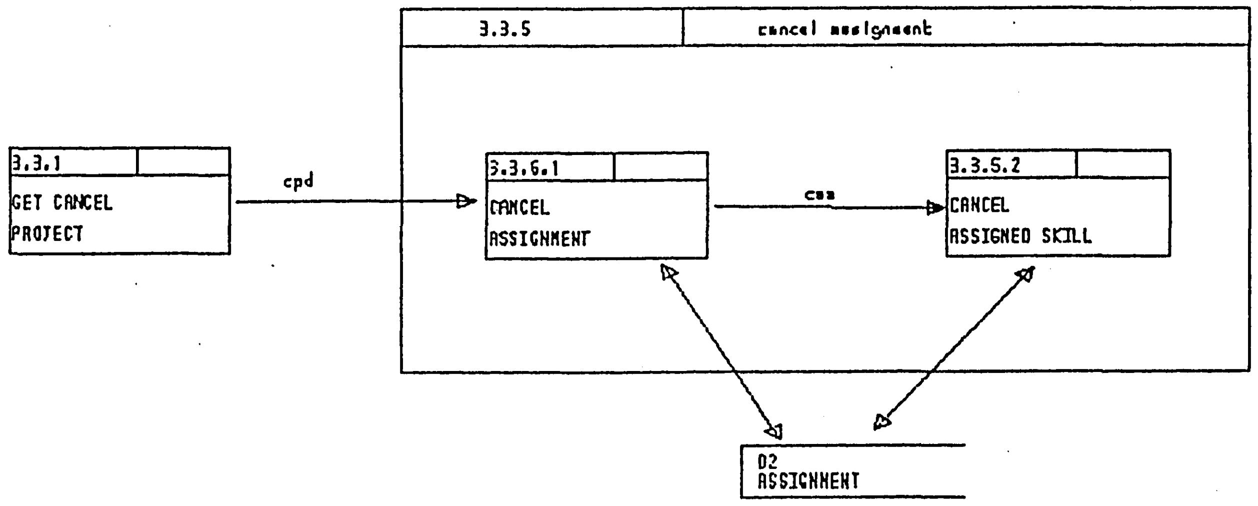

3.3.5
cancel assignment
3.3.1
GET CANCEL
PROJECT
cpd
3.3.6.1
CANCEL
ASSIGNMENT
csa
3.3.5.2
CANCEL
ASSIGNED SKILL
D2
ASSIGNMENT

Figure 8.3.14

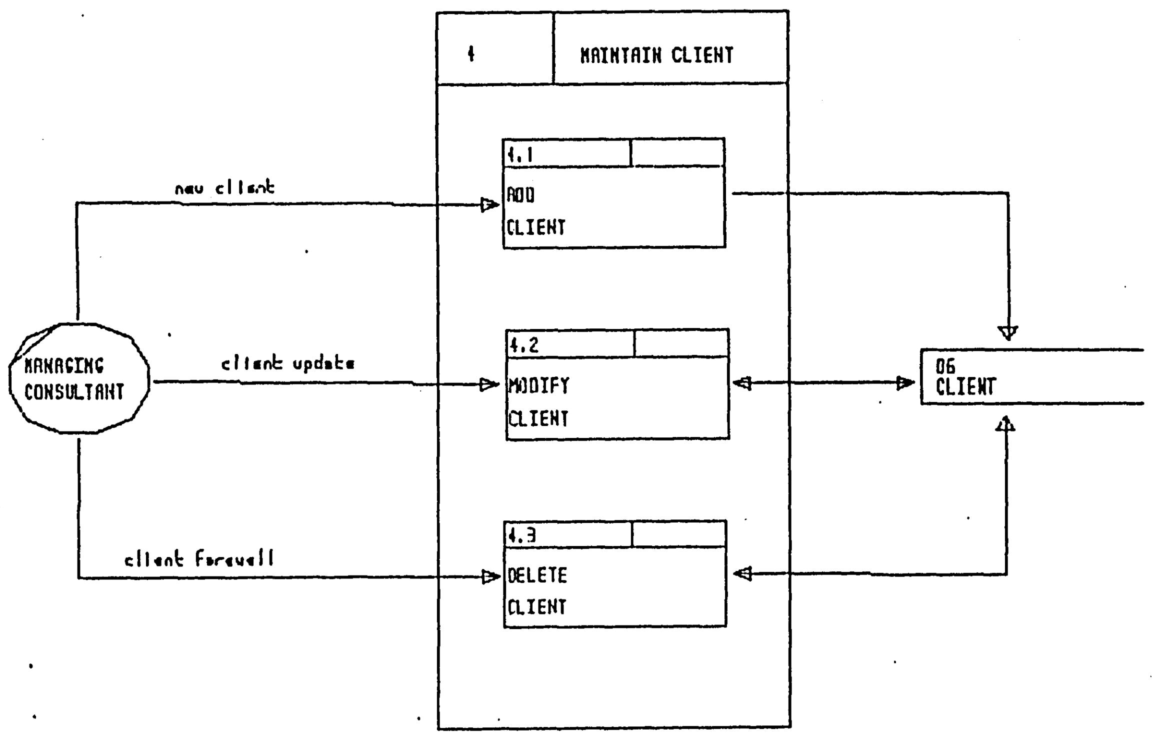

4
MAINTAIN CLIENT
4.1
ADD
CLIENT
4.2
MODIFY
CLIENT
4.3
DELETE
CLIENT
MANAGING
CONSULTANT
new client
client update
client farewell
06
CLIENT

Figure 8.3.15

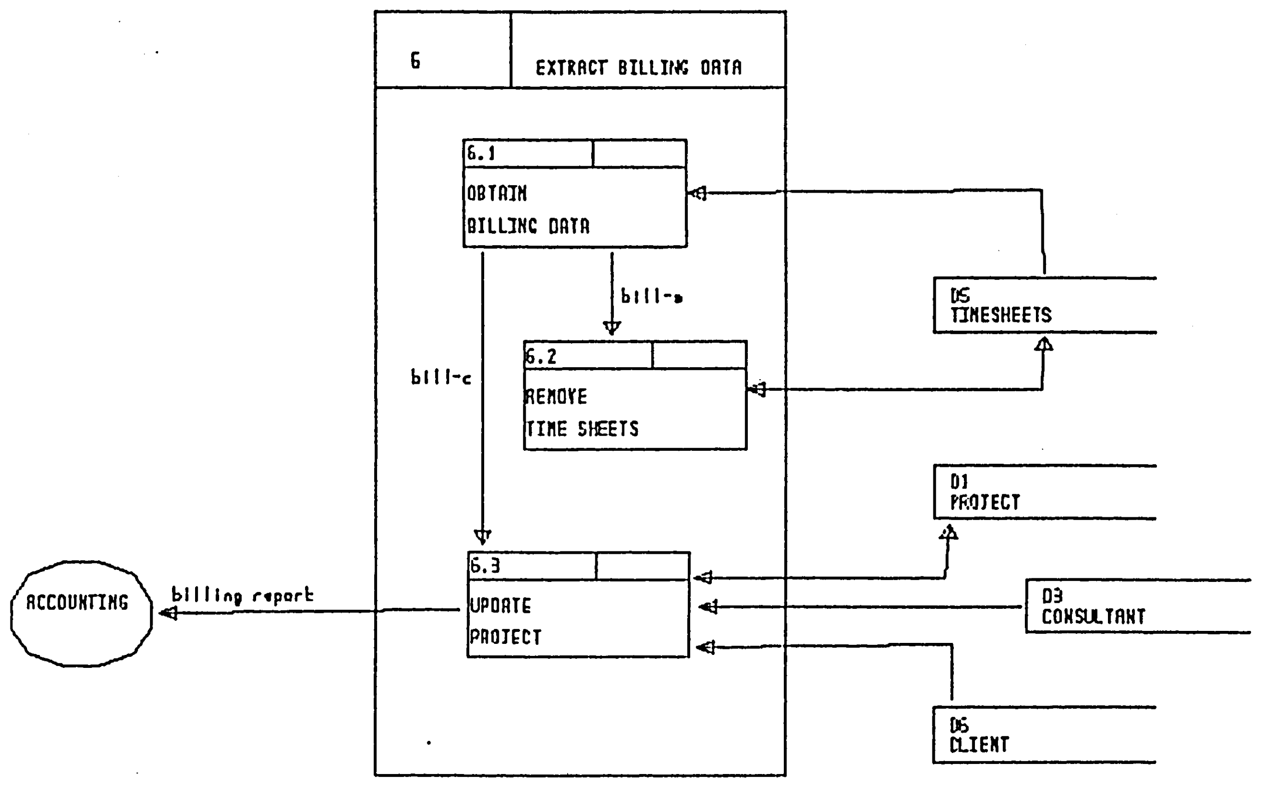

6
EXTRACT BILLING DATA
6.1
OBTAIN
BILLING DATA
bill-a
bill-c
6.2
REMOVE
TIME SHEETS
6.3
UPDATE
PROJECT
ACCOUNTING
billing report
D5
TIMESHEETS
D1
PROJECT
D3
CONSULTANT
D6
CLIENT

Figure 8.3.16

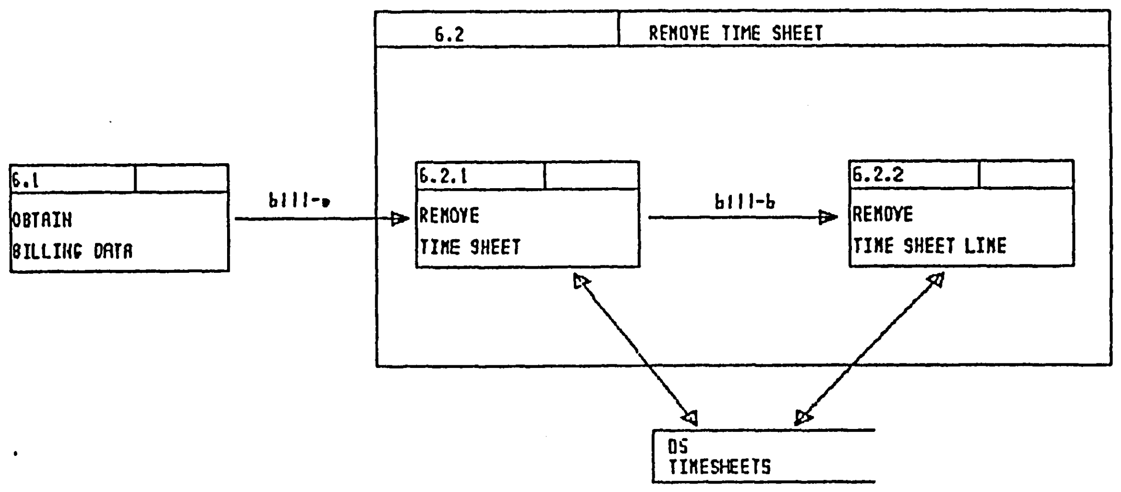

6.2
REMOVE TIME SHEET
6.1
OBTAIN
BILLING DATA
bill-a
6.2.1
REMOVE
TIME SHEET
bill-b
6.2.2
REMOVE
TIME SHEET LINE
DS
TIMESHEETS

Figure 8.3.17

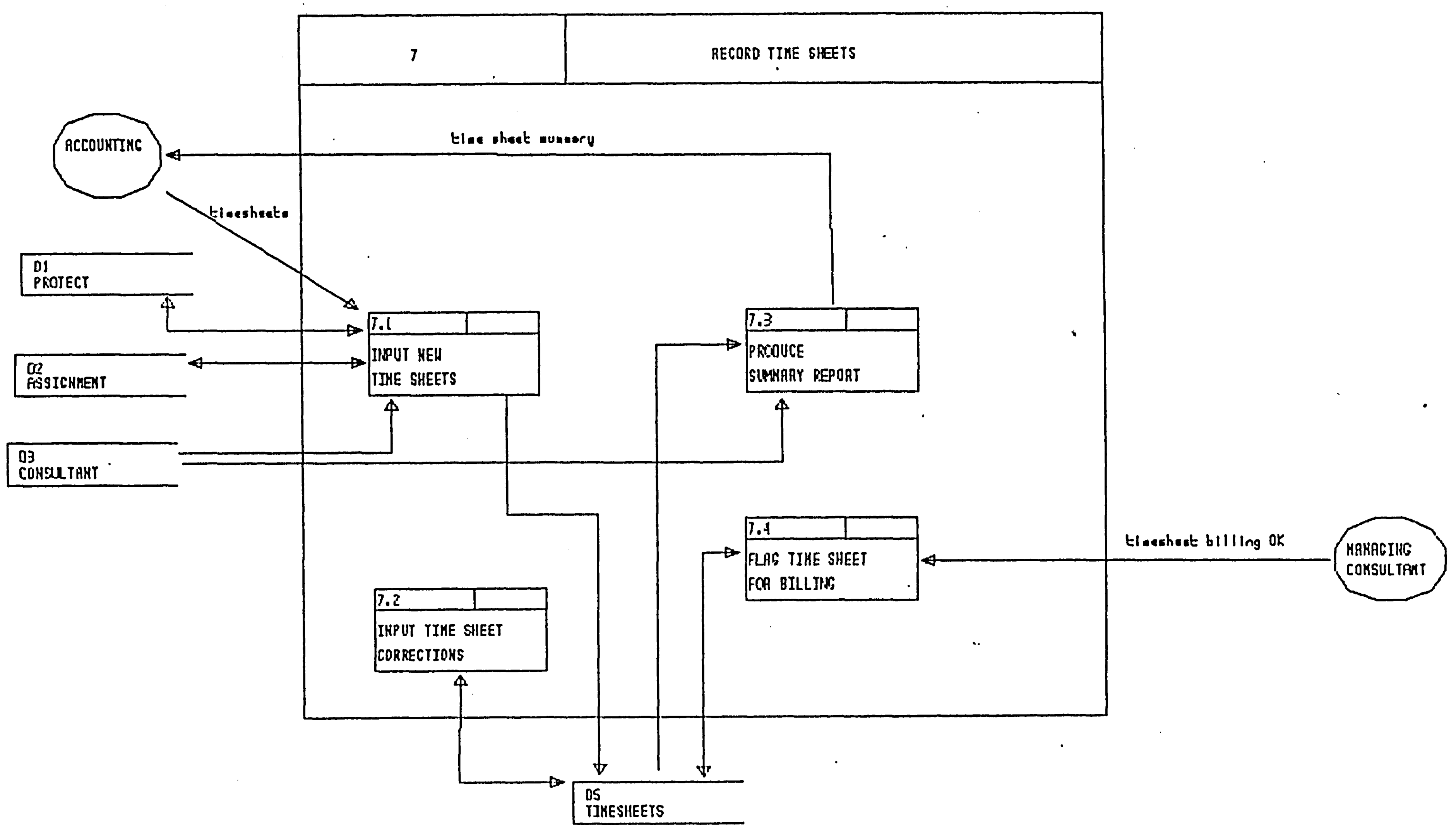

7
RECORD TIME SHEETS
ACCOUNTING
time sheet summary
timesheets
D1 PROTECT
D2 ASSIGNMENT
D3 CONSULTANT
7.1 INPUT NEW TIME SHEETS
7.3 PRODUCE SUMMARY REPORT
7.4 FLAG TIME SHEET FOR BILLING
7.2 INPUT TIME SHEET CORRECTIONS
timesheet billing OK
MANAGING CONSULTANT
D5 TIMESHEETS

Figure 8.3.18

8.5 The Entity Event Matrix.

The entity event matrix is shown in figure 8.4.

	1.1	1.2	1.3	2.1	2.2	2.3	3.1	3.2	3.3	4.1	4.2	4.3	6.1	6.2	6.3	7.1	7.2	7.3	7.4
PROJECT				R		R	I	M	M						M	M			
ASSIGNMENT			I/M	I	R	R			M							M			
CONSULTANT	I	M	M	R	R										R	R			
TIMESHEET														M		I			
TIMESHEET LINE												R	M			I		R	M
CLIENT							M		M	I	M	M			R				
ACQ. SKILL	M	I/M	M																
ASS. SKILL			I/M	I					M										
REQ. SKILL							I	I/M	M										

Figure 8.4

8.6 The Entity Life Histories

There are 9 entity life histories, one for each entity. Each event is annotated with its entity event diagram reference and each place with its ELH status.

The housekeeping function was not shown in detail on the entity event diagram. It is simply shown as function 5 on all the ELH.

The ELH diagrams are shown in figure 8.5.

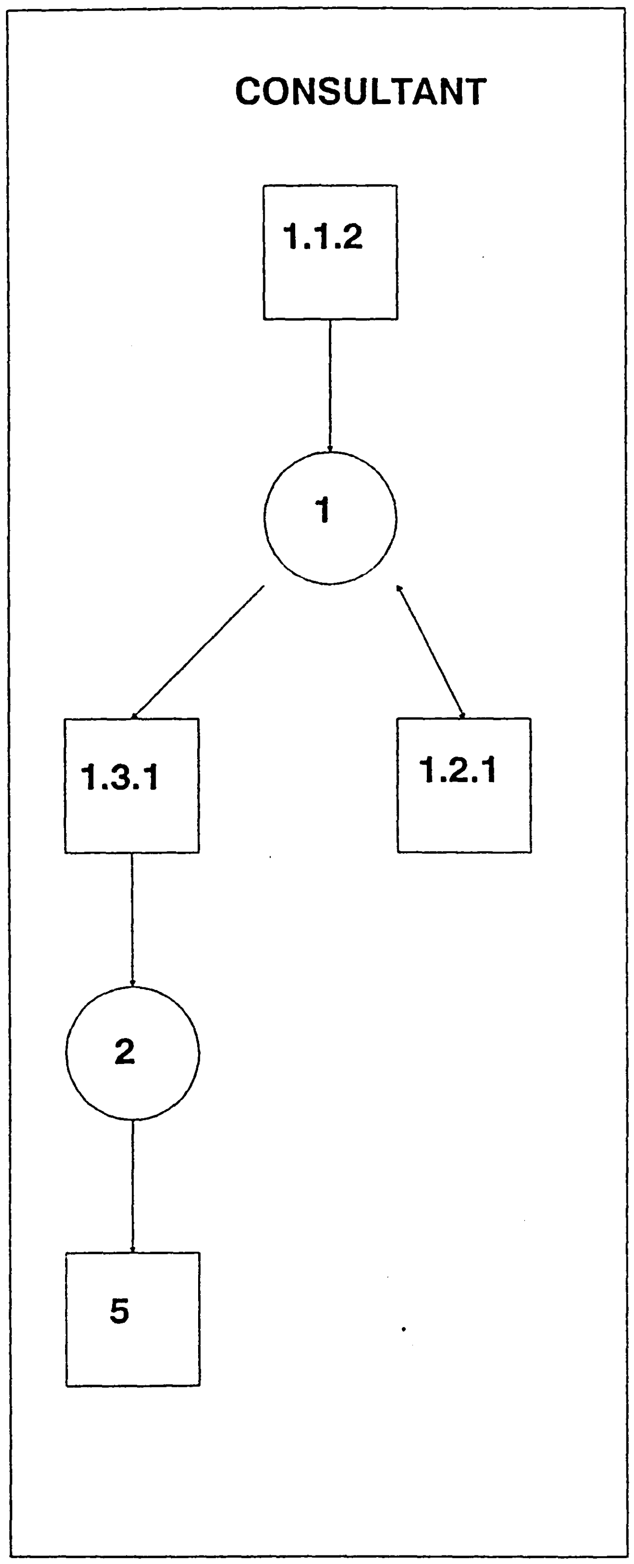


Figure 8.5.1

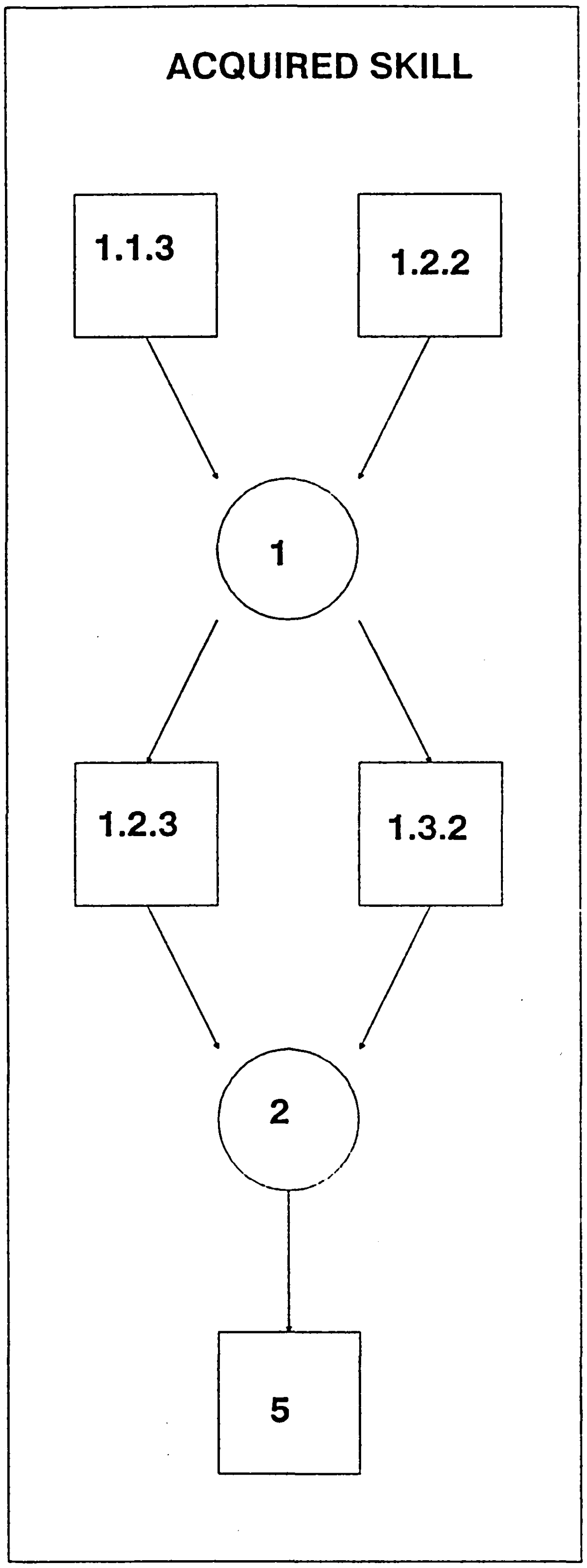


Figure 8.5.2

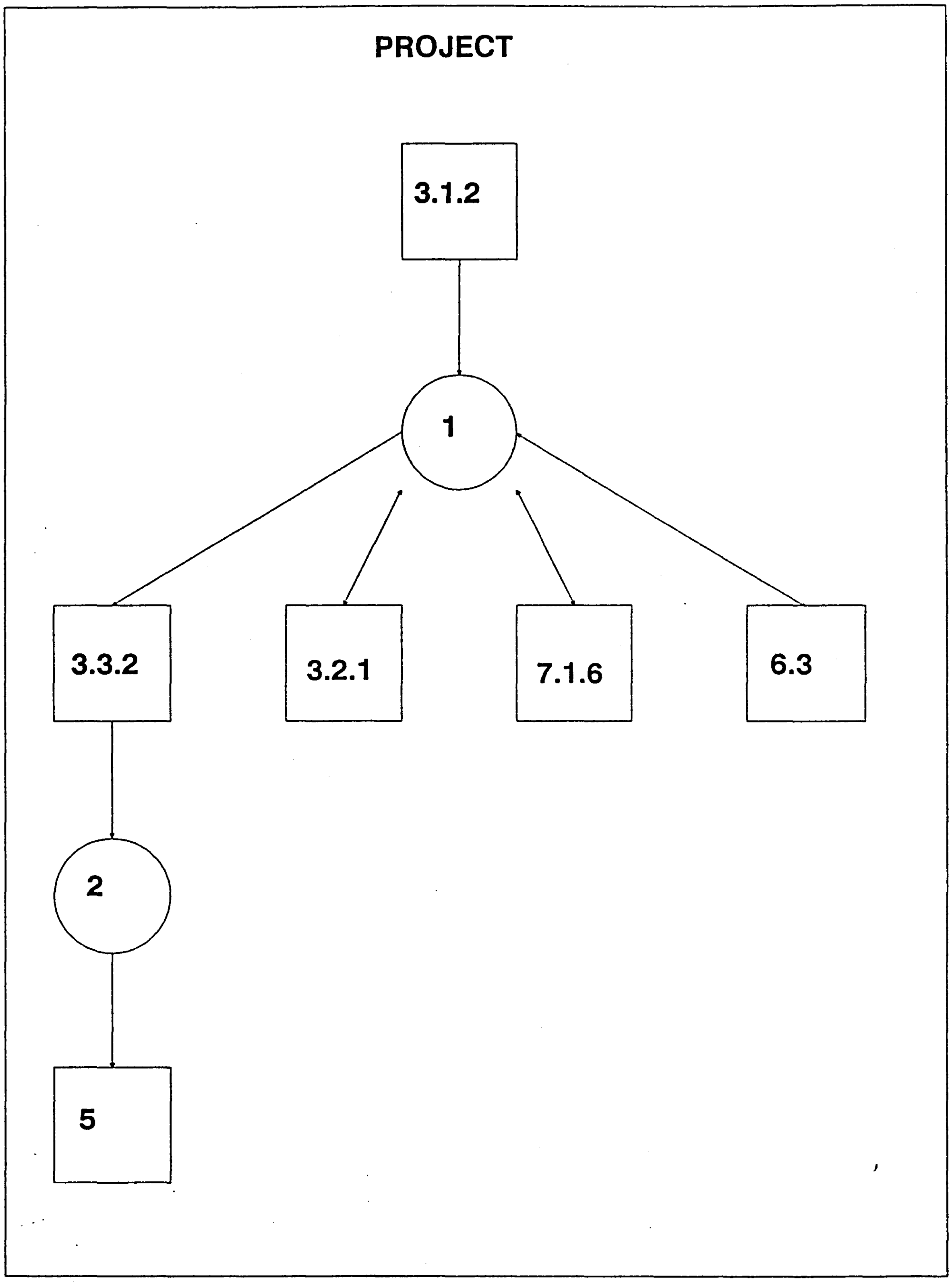


Figure 8.5.3

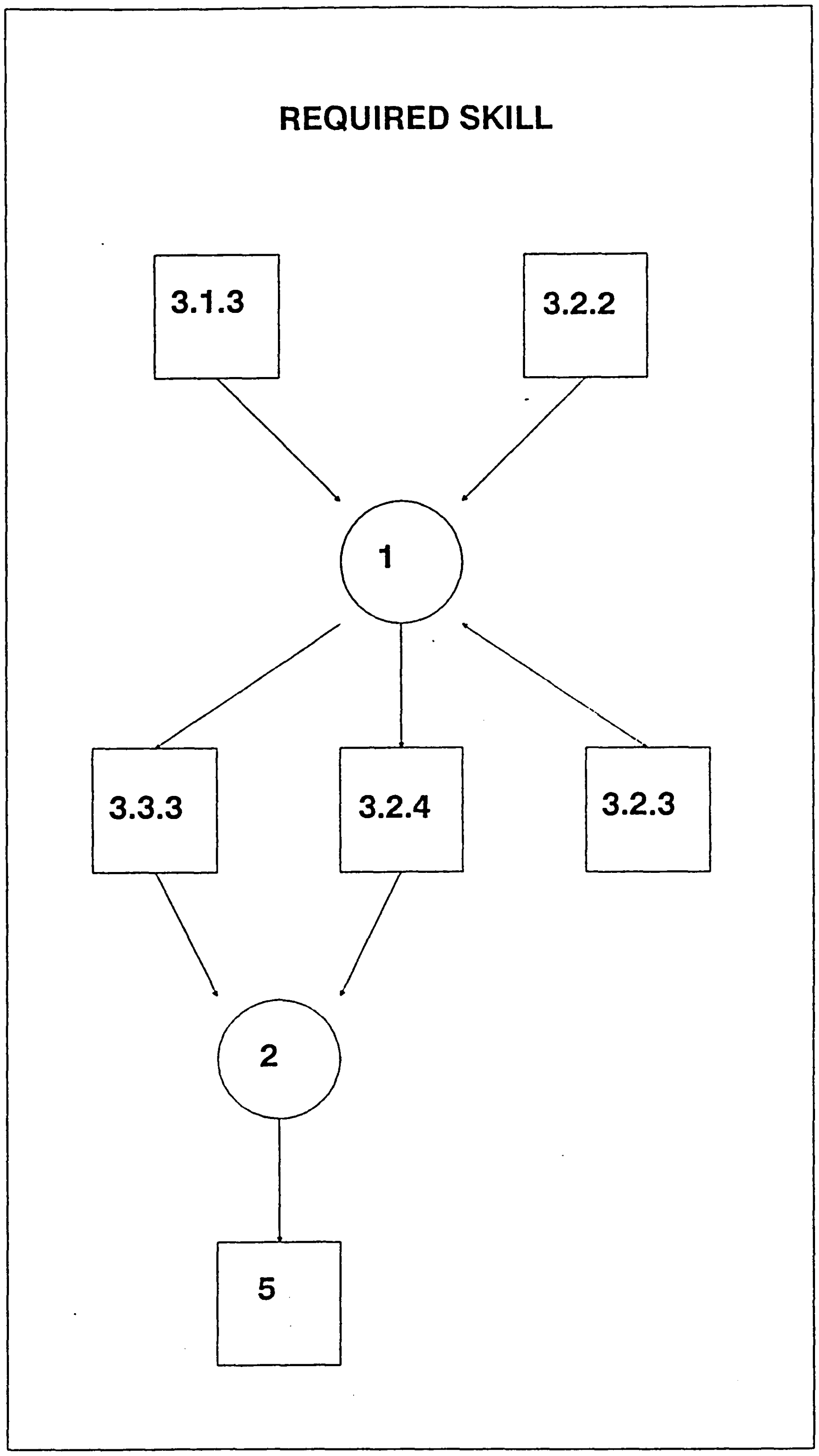


Figure 8.5.4

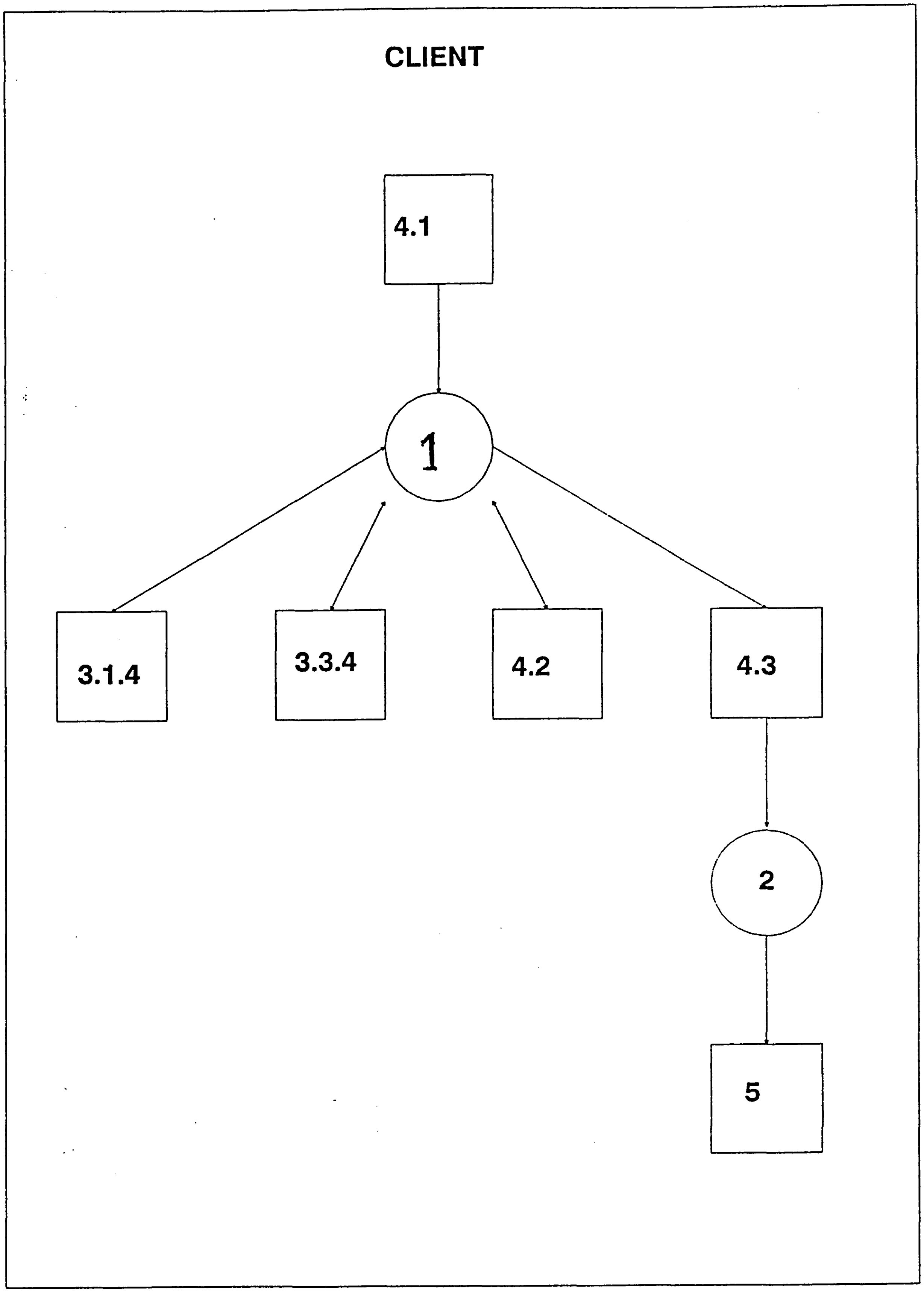


Figure 8.5.5

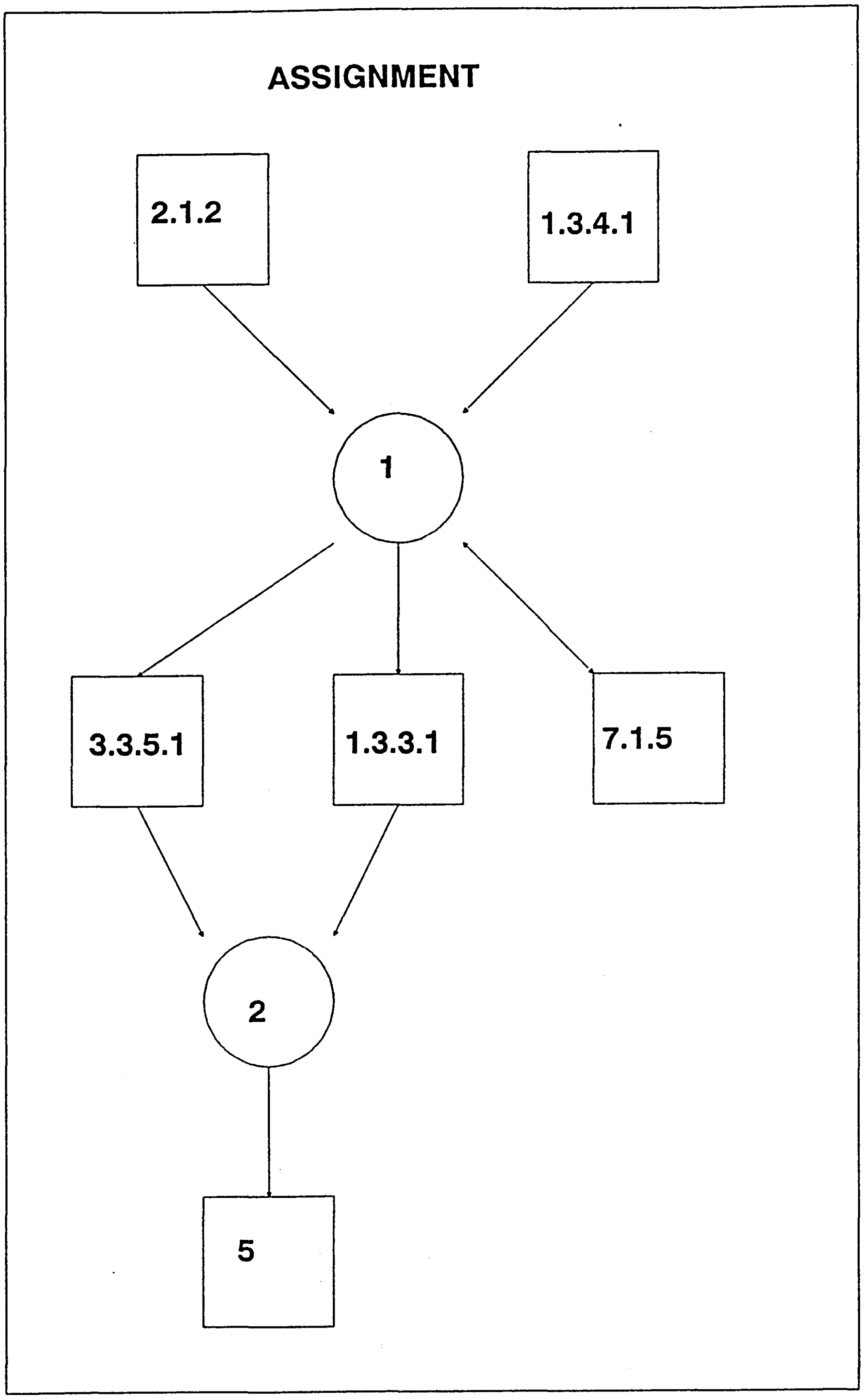


Figure 8.5.6

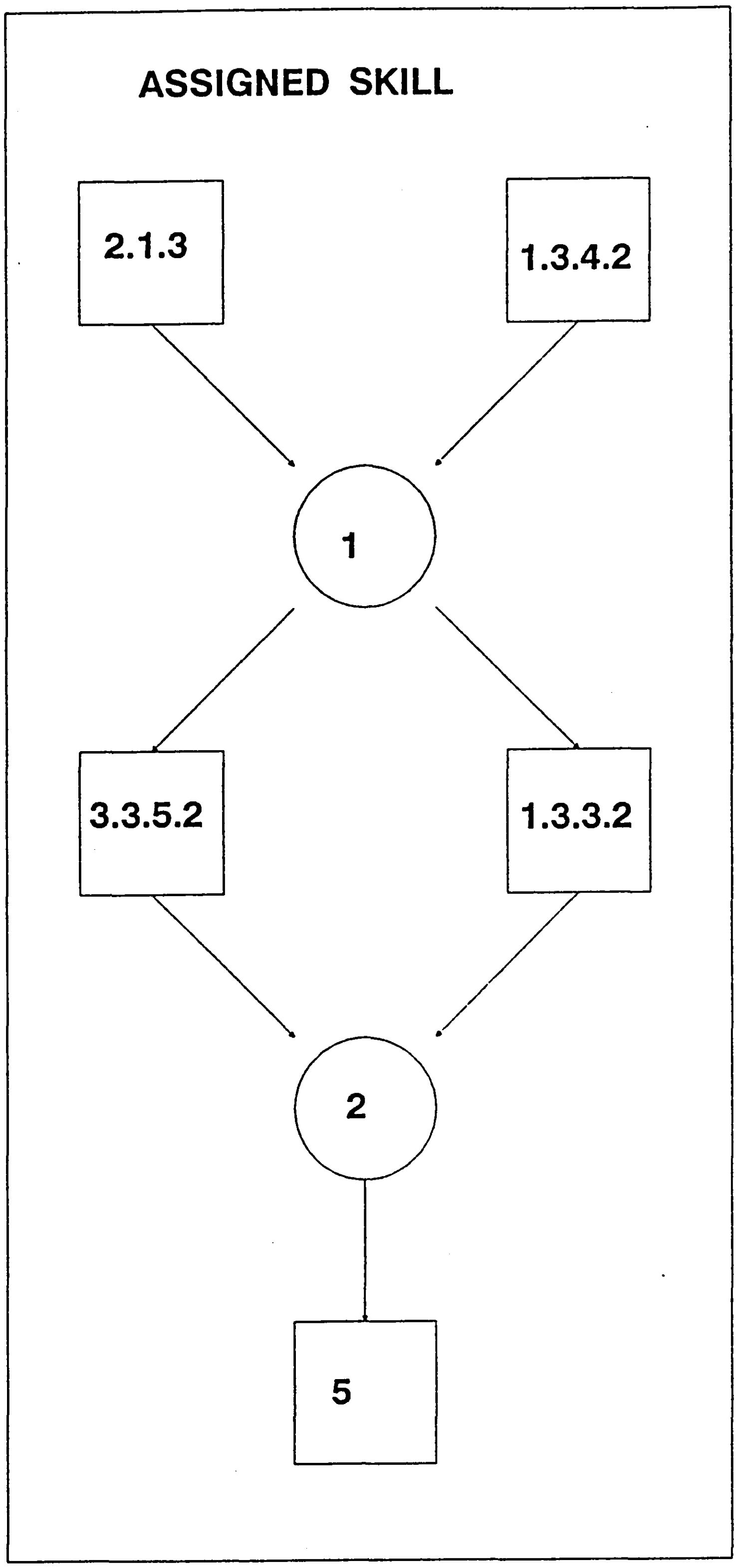


Figure 8.5.7

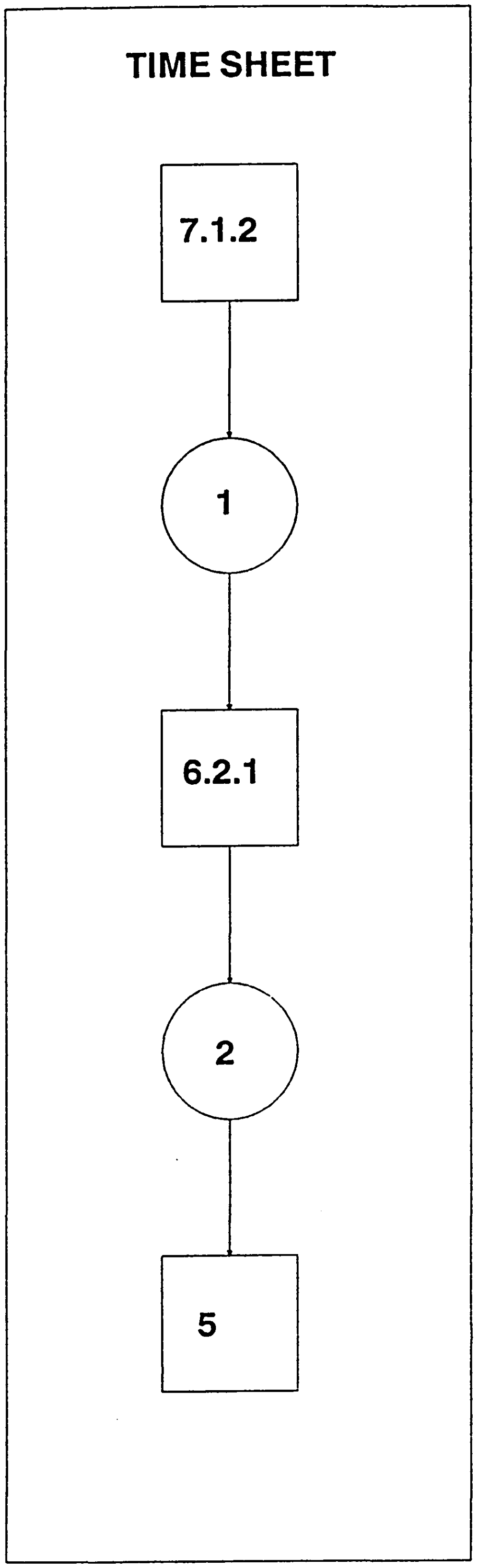


Figure 8.5.8

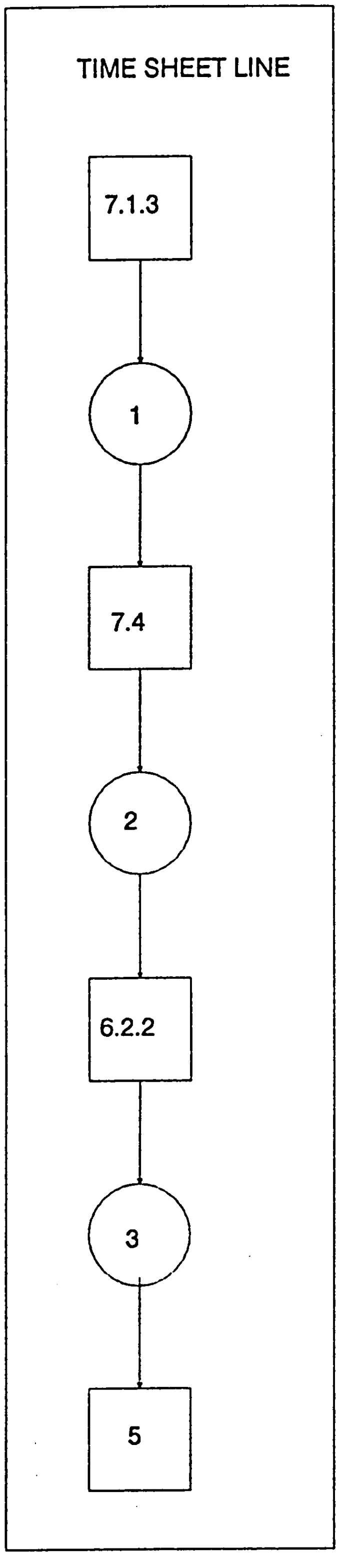


Figure 8.5.9

8.7 The Coloured Petri net

Figure 8.6 shows the coloured Petri net.

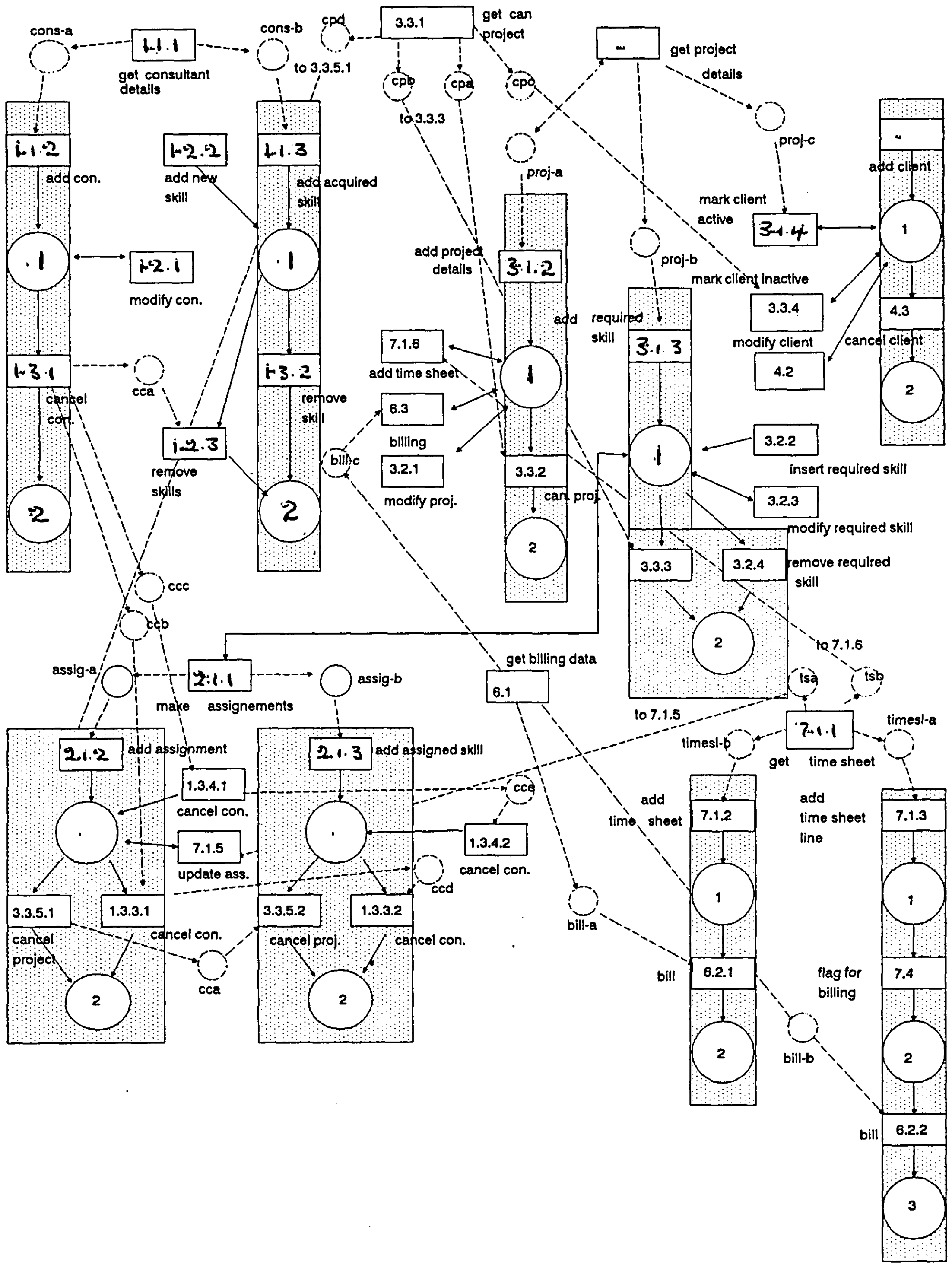


Figure 8.6

8.8 The Implementation

8.8.1 The System Implemented

A subset of the complete system was implemented comprising 18 events, 9 entities as relational tables and eleven message buffers also implemented as relational tables. Figure 8.7 shows the fragment of the system which was implemented.

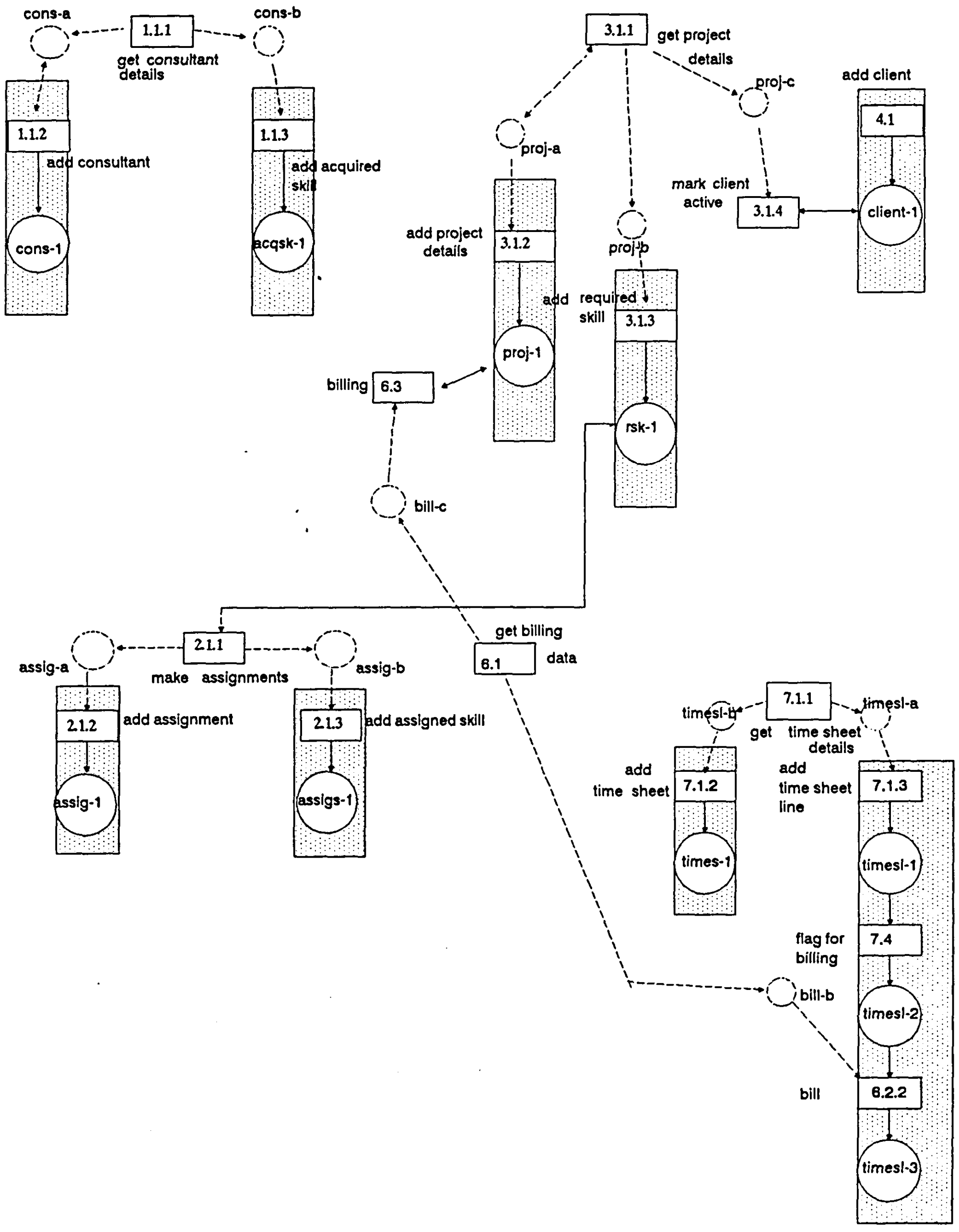


Figure 8.7

The application development environment was used linked to dBASE as the vehicle for table creation and dBASE linked to the Clipper compiler was used for program creation. Test and process programs were written for each transition.

The program listings are included as Appendix A. There were no major problems encountered during the implementation phase.

8.8.2 System Testing

The system was tested using the simulator as follows.

1. All tables were empty and a first test / execute cycle executed.

Programs 1.1.1, 3.1.1, 4.1, and 7.1.1 were enabled. These programs input details of new consultants, new projects, new clients and new time sheets.

1.1.1 Two new consultants were entered with their acquired skills.

Person#	1	2
Name	Cutts	Jones
Bill Rate	10	20
Skill #s	1,2	2,3

3.1.1 Two new projects were entered with their required skills.

Client#	1	2
Project#	1	2
Required skill	1,3	2

4.1 Two new clients were entered.

client#	1	2
name	SCP	LBMS

7.1.1 No new time sheets were entered.

Tables cons-a, cons-b, proj-a, proj-b, proj-c and client-1 now contain appropriate tuples which mark the net.

2. Test / process cycle 2

Programs 1.1.1, 1.1.2, 1.1.3, 3.1.1, 3.1.2, 3.1.3, 3.1.4, 4.1 and 7.1 were enabled.

1.1.1 No new consultants were entered.

1.1.2 Consultant details were added to the database.

1.1.3 Acquired skill details were added to the database.

3.1.1 No new projects were entered.

3.1.2 Project details were added to the data base.

3.1.3 Required skill details were added to the database.

3.1.4 Clients 1 and 2 were marked active.

4.1 No new clients were entered.

7.1.1 No new time sheets were entered.

This cycle processed all the messages in the system to deposit tuples in cons-1, acqsk-1, proj-1, rsk-1 and to update tuples in client-1.

3. Test / process cycle 3

The input programs 1.1.1, 3.1.1, 4.1 and 7.1.1 were enable together with program 2.1.1 the program which makes assignments.

No new data was entered.

2.1.1 The following assignments were made.

client#	project#	skill#	person#	assign Y/N
1	1	1	1	Y
1	1	3	2	Y
2	2	2	1	N
2	2	2	2	Y

All three required skills have been assigned to consultants. Message buffers assig-a and assig-b now contain tuples relating to the assignments.

4. Test / process cycle 4

The input programs were enabled together with programs 2.1.2 and 2.1.3.

No new consultants, clients or projects were entered.

2.1.2 Assignments were added to the database.

2.1.3 Assigned skills were added to the database.

7.1.1 Two time sheets were entered.

person#	client#	project#	hours this week
1	1	1	10
2	2	2	20

Message buffers timesl-a and timesl-b now contain tuples.

5. Test / process cycle 5

The input programs were enabled together with programs 7.1.2 and 7.1.3.

No new input was entered.

7.1.2 Time sheets were added to the database.

7.1.3 Time sheet lines were added to the database.

Tables times-1 and timesl-1 now contain tuples.

6. Test / process cycle 6

In addition to the input programs program 7.4 was enables as time sheet lines were available for billing.

7.4 The following time sheet lines were made available for billing.

person#	client#	project#	Bill Y/N
1	1	1	Y
2	2	2	Y

Table timesl-2 now contains tuples.

7. Test / process cycle 7

In addition to the input programs 6.1 was enabled.

6.1 Billing data was extracted and written to message buffer bill-c for billing and project update and to bill-b to remove billed time sheet lines from the database.

8. Test / process cycle 8

Again the input programs were enabled together with programs 6.2.2 and 6.3.

6.2.2 Billed time sheet lines were marked for removal from the database.

6.3 The project tuples were updated with the hours billed this week.

The eighth cycle concluded the processing until further input is made. The only programs now enables are the input programs.

All the message buffers are empty.

Database tables contain data as follows.

cons-1	contains 2 tuples
acqsk-1	4
client-1	2
proj-1	2
rsk-1	3
assig-1	3
assigs-1	3
times-1	2
timesl-1	empty
timesl-2	empty
timesl-3	2

8.8.3 The Test Programs Revisited

The enabling rules for each transition are implemented in the test program associated with each transition or process. The test program code is included as Appendix B.

The following transition have test programs which always return true since there are no guards.

Programs 1.1.1, 3.1.1, 4.1 and 7.1.1, the input programs.

Throughout the execution input was always possible.

The guards on the other programs are as follows.

Test 1.1.1 *enabled if a tuple (token) exists in*	cons-a.
1.1.3 "	cons-b.
3.1.2 "	proj-a.
3.1.3 "	proj-b.
3.1.4 "	proj-c.
2.1.1 *enabled if unassigned tuples in*	rsk-1.
2.1.2 enabled if a tuple (token) exists in	assig-a.
2.1.3 "	assig-b.
7.1.2 "	timesl-b.
7.1.3 "	timesl-a.
7.4 "	timesl-1.
6.1 "	timesl-2.
6.2.2 "	bill-b.
6.3 "	bill-c.

The test/process cycles and enabling of processes followed the above rules.

Chapter 9 CONCLUSIONS

9.1 Research Focus

The research focussed on modelling, verification and simulation within a development approach to parallel information systems. The research sought to include the areas of responsibility for information engineers, construction of specifications, development of designs and the implementation of information systems. The idea that each of these areas of responsibility could be represented in some notation or notations was adopted together with the need to convert from notation to notation during the development process.

The 'real' world of information systems is parallel. The requirement to engineer parallel information systems together with the motives of increased performance, decreased costs, flexibility of configuration, reliability and ease of development was the driving force for the research.

There is a need to bring new and developing techniques and tools into a methodological framework which embraces modelling, verification and simulation. To achieve parallelism loosely coupled systems were chosen. In this type of system, programs execute on independent processors to achieve the goal of information system.

Many of the requirements could be obtained from the many well developed structured systems analysis and design methodologies. In addition the research required a communications discipline on asynchronous processes based on local conditions. Petri nets provided a modelling and analysis approach for this discipline. The use of Petri nets also contributed to solving the problems of synchronisation and deadlock and Coloured Petri nets with their high level of modelling power contributed to the modelling and analysis of complete information systems.

The hypothesis developed in the early days of the research centred on four areas of concern, parallelism in information systems, modelling and analysis of dynamic states, simulation and prototyping, and the pragmatic application of theory. The main thrust of the research has been to concentrate on the techniques. The use of Place Transition nets for Entity Life Histories and Coloured Petri nets for system modelling proved to be excellent notations for the modelling and analysis of dynamic states in parallel information systems. The development process, developing Entity Event Diagrams as a restriction of Data Flow Diagrams and the transformation of Entity Event Diagrams and Entity Life Histories into Coloured Petri nets proved easy to use and highly beneficial to the building of 'correct' System nets. The process embraced parallelism, prototyping, simulation, user involvement whilst remaining a very practical approach. The development environment enabled a full case system to be developed easily and rapidly demonstrating many of the benefits that the approach and techniques offered.

9.2 Overview

The work clearly demonstrates that the approach, techniques and notations embrace the notions of parallelism in a formal way which allows for specification, analysis of the specification, design and realisation. To develop an information system which comprises objects which exist independently and concurrently and which are distributed with interaction via messages then the method must provide for each object to be identified, specified, analysed and developed independently with the knowledge that the objects can be brought together to create the application without the loss of any of the properties of the independent development. The approach developed is very strong in its formal representation of events and the dynamic states which trigger events, these dynamic states allow the construction of prototypes since the application can be built in sections due to the property it possesses of 'correct by construction'. This property was used in the development of the case study.

Information systems built in this way exhibit high cohesion with strong links from the environment to the schema as well as low coupling. This property will lead to easier maintenance and modification. The information system can accept new components 'plugged in' at all stages of development and operation. Again this property was demonstrated in the development of the case study. An approach which develops small scale prototypes coupled to synthesis of the application leads to flexible, low cost development which can be highly adaptive to user requirements and effectively user driven. The application can be synthesised from components, the components being the objects or entities. Further the method provides for a wide view of the information system's environment. Because of the incremental development features of the approach, the building of prototypes and the inherent parallelism featuring the environmental dimensions of structure, geography, time, activity, space, economy and social aspects are easily accommodated.

The approach developed covers information system inception to realisation using transformations from one notation to another as the mechanism. In many instances the transformations have been formalised and mechanised. The approach incorporates a number of fundamental tenets including modelling, functional decomposition, encapsulation, prototyping, verification and simulation. Finally the approach is supported by a number of techniques and integrated tools.

Many of the techniques are familiar, some have been developed from an existing technique, several are new. Entity event diagrams are a useful development of data flow diagrams, they are a more rigorous form of data flow diagram in which data stores must be entities and processes must be independent events which effect the entities. The research derived steps to develop entity event diagrams from data flow diagrams. One of the major tasks is the development of entity life histories. Entity life histories are fundamental to the approach. The application and theory of Petri nets was used to develop network entity life histories which may be represented formally. In this fashion entity life histories can be proved to be complete and consistent. This idea is one of the fundamental development concepts of the thesis enabling formal evaluation of the application design before simulation. The use of Petri nets was developed during the research and subsequently applied as part of the applied research by a software house, a pharmaceutical manufacturer and an international bank. The formal transformations from a set of proven entity life histories and an entity event diagram into the Coloured Petri net is again one of the fundamental development concepts. The transformation rules have been specified, used on the case study and partly automated.

A number of tools proved very useful. Auto-Mate plus from LBMS was used for the production and checking of diagrams. However, Auto- Mate plus did not provide an entity life history facility. An appropriate tool was developed which comprised an editor and an analyser. This tool is being enhanced but has now been overtaken by commercially available tools especially the Coloured Petri net system from Meta Software. The major tool developed was the application development environment. The tool provides a System net editor, a database create, modify and view facility, a program development facility, a program test environment, a system test environment and an application simulation environment. The environment is that of Coloured Petri nets.

All of the above was demonstrated by case work using a case study supplied by Learmonth and Burchett Management Systems.

9.3 The Approach

The approach should be regarded as a framework, a framework which encourages good practice and encompasses an engineering discipline. Good practice through a disciplined approach to development and the use of appropriate techniques and tools. Good practice in that the approach and the production of documentation are integrated. These are the good practices extracted from the structured development paradigm and from the application and theory of Petri nets. The linking of these two areas proved to be ideal for the development of parallel information systems.

An engineering discipline is engendered through the use of graphical and formal notations as appropriate together with transformation rules to move between notations. In this way rigour was included which was required for many of the assertions such as correct by construction and freedom from deadlock. The transformation rules furnished the method of providing a unifying approach required. The approach moves smoothly from data flow diagrams to entity event diagrams to entity life histories and Coloured Petri nets and on into System nets used for prototyping, construction and simulation of parallel information systems. The approach clearly separates modelling and logical representation from physical realisation. This solves the problem discovered early in the research, which is the direct coupling of thinking about concurrency to realisation.

9.4 The Techniques

9.4.1 Entity Event Diagrams

The idea to develop data flow diagrams originated from a need to discover atomic events and entities which could be distributed across processors. A further benefit resulted in ease of development. Modules which effect a single entity, are atomic, are very highly cohesive in that they handle a single event and are only coupled via message passing are easy to develop, test, modify and maintain. Side effects are eliminated and testing is easy as full-blown testing is truly integrated. The guidelines for developing entity event diagrams proved successful and the benefits were realised.

9.4.2 Place Transition Nets for Entity Life Histories

Place Transition nets proved to be ideal for development and analysis of Entity Life

Histories (ELH). Development used the standard structures of sequence, selection and iteration. However, Petri nets provided two further structures not available naturally in other notations, they are parallel and alternate, both necessary for ELH design. ELH as Petri net graphs provided a very useful interface with users so much that they are now used in industry and on courses. The token game allowing animation of the ELH is particularly beneficial to enable developers and users to visualise the dynamic behaviour.

The mathematical equivalent notation for Petri nets enabled a number of properties to be discovered. It is necessary to ensure all ELH are free from deadlock and that is there is always a route from any state of an entity occurrence through to archive or deletion of the occurrence. Equally it is necessary to be able to identify a number of properties about an ELH, for example, that no entity occurrences are created or deleted from within the ELH itself. It is also necessary to identify traps, can an entity occurrence become trapped in a cycle, for example, a backorder, review, report cycle with no exit route into normal processing.

All of the above were proved using set theory and place invariants, which analyse the ELH structure and not its dynamic behaviour. This overcomes the major problem of state space explosion when analysing behaviour. Finally the use of transition invariants allows all of the routes from entry to deletion of an entity occurrences to be identified. This ensures the ELH meets the user's requirements. The above analyses are implemented in Petri net analysis packages.

9.4.3 System Modelling using Coloured Petri Nets

Entity Life Histories represent a single entity. Each place in an ELH represents the state of an entity occurrence, a single token can therefore be used to represent an occurrence. ELH therefore are adequately represented by Place Transition Nets.

The first transformation of ELH and EED into Coloured Petri nets involves the bringing together of the individual ELH into a single net. Clearly different places represent different entities. Tokens must now be individual. We could colour each token, red for the customer entity, blue for the order entity, however, it is better to use a token vector whose elements represent the attributes of the entity. The individual ELH are connected by message buffers again represented by places on the Coloured Petri net. Message buffers are individual so again individual tokens representing the message attributes are required. Finally message content may vary according to their processing by an event. The output of an event can be represented by functions in Coloured Petri nets.

These considerations led to the decision to use Coloured Petri nets. Sophisticated software now exists for the development and analysis of Coloured Petri nets. The package CPN-Palette from the Meta Software Corporation has been developed over the past two years. Coloured Petri nets with their net inscriptions and individual tokens allow the rigorous formality of ordinary Petri nets to be retained whilst enabling the size of models to be kept small. This enhances the graphical model building features but unfortunately very much complicates the mathematics for analysis. However the mathematical notation for Coloured Petri nets does allow similar properties to the ELH to be investigated. Questions such as, Will transactions as a sequence of messages always pass through the system? and What are all the routes a transaction can take? can be answered.

Entity occurrences and their life can be considered to be orthogonal to transactions as a series of messages. Entity occurrences originate at the top of the net and are archived at the bottom whilst transactions pass from left to right interfacing with the entity occurrences. Because of the way in which nets can be fused according to mathematical rules which preserve all of the net's properties then one of the research's objectives that of 'correct by construction' was achieved. These rules have been recently extended to develop the idea of hierarchies of nets.

9.4.4 The Development Environment

Prototyping, incremental development, 'correct by construction' were all proved using the case study and the development environment. The ability within the environment to build the System net incrementally alongside database construction and program writing was very successful. Indeed all testing was integrated with construction and due to the prior analysis the system worked without further activity. The case study system is working and demonstrable. The overall conclusion is that the research has contributed significantly in the areas of modelling, verification and simulation particularly applied to parallel information systems.

9.4.5 A Comparison with Other Techniques

The approach developed rules for the formal transformation of data flow diagrams into entity event diagrams, and the combination of entity event diagrams and entity life histories into coloured Petri nets. Commercially available methods do not address the issue of transformations, the use of an underlying common modelling technique based on Petri nets, particularly the formal notation for Petri nets allowed transformations to be developed and proved by their use on the case study.

Entity life histories are generally represented by Jackson structure diagrams or by LBMS's network convention. The use of Place Transition nets obviated the need for quit and resume markers. They modelled naturally state indicators, parallel events, alternate events as well as the standard constructs of sequence, selection and iteration. The formal mathematical notation allowed entity life histories to be analysed for deadlocks, traps and bottlenecks which is not possible using the heirarchical or network notations.

The use of Coloured Petri nets to model information systems allowed complex parallel information systems to be modelled using a single technique. This single technique is different in that a formal notation exists which facilitated formal verification. The verification of models of complex information systems is a major improvement over techniques such as data flow diagrams and logical data structures. Finally, modelling using constructs which naturally represent parallelism leads to the easy development of a simulation environment and further into implementation on parallel architectures. These properties were demonstrated in the development of the case study.

9.5 Commentary on Possible Future Paths

The major benefit of this approach is the ability to build small scale sub-systems which can be guaranteed to fit into the total system even before the total system is known. To realise this benefit a fully integrated set of tool is required.

Kurt Jensen, the developer of Coloured Petri nets, has spent part of the last few years on secondment to the Meta Software Corporation, Cambridge, USA. Meta have developed the CPN-Palette comprising a CPN Editor which supports hierarchy and fusion coupled to standard analysis support for Coloured Petri nets. Meta have used the software to describe systems such as telephone systems, protocols, buffer systems and VLSI chips. Apart from a very small attempt to link SADT to CPN all of the above work uses hierarchical CPN directly as the model for development. Meta have also extended Standard ML to enable colour sets and CPN variables to be declared. CPN ML is then used to calculate enabling and the occurrence of bindings. This effectively implements the CPN specification.

The potential for Coloured Petri nets is very well supported by this package. To fully realise this potential much more work is necessary to make CPN available and acceptable to the potential users. Translations from ideas familiar to system professionals into CPN are required.

Further the rather theoretical work on stochastic and timed Petri nets has been very recently added to CPN. This will enable timing to be added to the CPN so that queue sizes and bottlenecks can be established. The answers to these questions would allow the dynamic reconfiguration of processors in a parallel system to satisfy the demands of the application.

Generally the theoretical work is more than adequate the concentration must be on the application of the work.

REFERENCES

Ajmone Marsan M, Balbo G, Conte G, *A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multi-Processor Systems,* ACM Transactions on Computer Systems Vol 2, No. 2,(1984)

Ashworth C, Goodland M, *SSADM: A Practical Approach,* McGraw-Hill, (1990)

Aspinall D, Dagless E L, Dowsing R D, *Design Methods for Digital Systems Including Parallelism*, IEEE Journal Electronic Circ. & Syst. Vol 1. No. 2,(1977)

Avison D E, Fitzgerald G, *Information Systems Development, Methodologies, Techniques and Tools*, Blackwell Scientific Publications.(1988)

Burns A, Rathwell M A, *A Communications Environment for Co-operative Information Systems Development,* Software Engineering Journal,(1987)

Chung W H, Ryoung H, Park K H, Kim M, *A Transformation of Timed Petri Nets for Response Time Estimation,* Private Communication - Korea Advanced Institute,(1988)

Cohen B, *Justification of Formal Methods for System Specification,* Software and Microsystems. Vol. 1, No. 5,(1982)

Cohen B, *A Rejustification of Formal Notations*, Software Engineering Journal. Vol. 4, No. 1, (1989)

Commoner F, Holt A W, Even S, Pnueli A, *Marked Directed Graphs,* Journal of Computer and System Science, 5,(1971)

Commoner F, *Deadlocks in Petri Nets*, Report, Applied Data Inc,(1972)

Coolahan J E, Roussopoulos N, *A Timed Petri Net Methodology for Specifying Real Time System Timings,* International Conference on Timed Petri Nets, Torino,(1985)

Crookes D, *Translation as a Language Implementation Technique for Supercomputers*, Software Engineering Journal Vol. 3, No. 2, (1988)

Cutts G, *Structured Systems Analysis and Design Methodology*, Paradigm Publications(1987)

Cutts G, Bidwell L, *Masters Project (Bidwell) implementing the work of Taubner - 82,* SCP Library (1997)

Cutts G, *Systems Engineering and Nets,* 9th European Workshop on Application and Theory of Petri Nets, Venice, Italy,(1988)

Cutts G, *Structured Systems Analysis and Design Methodology Techniques and LSDM*, First European Conference On IT for Organisational Systems, EURINFO,(1988) Cutts G, Onley M, *Masters Project (Onley), Parallelisation of mathematical algorithms,* SCP Library (1988)

Cutts G, Magee D, *A Net Based Application Design Environment and Tool for Simulation of Distributed and Parallel Systems*, Pratique Des Methodes, Universite De Nantes,(1989)

Cutts G, *Structured Systems Analysis and Design Methodology, Second Edition*, Blackwell Scientific Publications, (1991)

Cutts G, *Proving PLC Programs with Petri Nets*, 13th International Conference on Application and Theory of Petri Nets, (1992)

Cutts G, *Modelling and Analysis of Entity Life Histories with Petri Nets*, IFIP -92 (1992)

De-Bondeli P, *Models for the Control on Concurrency in ADA Based on Predicate Transition Nets,* ADA- Europe, (1983)

De-Marco T, *Structured Analysis and System Specification*, Yourdon Press, (1978)

Diaz M, *Modelling and Analysis of Communications and Cooperation Protocols Using Petri Net Based Models*, Computer Networks, (1983)

Diaz M, *Applying Petri Net Based Models in the Design of Systems*, Advances in Petri Nets, (1987)

Downs E, Clare P, Coe I, *Structured Systems Analysis and Design Method: Application and Context,* Prentice Hall (1988)

Dugan J B, *Extended Stochastic Petri Nets: Application and Analysis*, Ph, D. Thesis, Duke University, (1984)

Finkelstein C, *An Introduction to Information Engineering: From Strategic Planning to Information System*, Addison Wesley, (1989)

Fitzgerald G, *Information Systems Development for Changing Environments,* First European Conference on IT for Organisational Systems, EURINFO, (1988)

Florin G, Natkin S, *Evaluation based upon Stochastic Petri Nets*, Informatik Fachberichte, Springer Verlag, (1981)

Foulk P W, *Directed-Graph Models and their Application to Software Development,* Software and Microsystems, Vol. 1, No. 7, (1982)

Fritzson P, *Towards a Distributed Programming Environment Based Upon Incremental Compilation,* Ph. D. Thesis, Linkoping University, Sweden, (1984)

Gain C & Sarson T, *Structured Systems Analysis: Tools and Techniques*, Prentice Hall, (1979)

Gallard R H, *An Extension in the Definition of a Petri Net Execution*, The Computer Journal, Vol 30, No 1, (1987) Genrich H J, *Predicate Transition Nets,* Advances in Petri Nets Vol 1, (1986)

Genrich H J, *A Formalism for Higher Level Petri Nets,* GMD, St. Augustin, Germany, (1988)

Genrich H J, Lautenbach K, *System Modelling with High Level Petri Nets,* Theoretical Computer Science, 13, (1981)

Gorton I, *Parallel Program Design using High-level Petri Nets,* Concurrency: Practice and Experience (1993)

Ghezzi C, Mandrioli D, Pezze M, *Petri Nets as a Support to Symbolic Execution of Concurrent Ada Programs,* Private Communication - Politecnico di Milano, (1988)

Guttag J V, Horowitz E, Musser D R, *The Design of Data Type Specifications, Current Trends in Programming Methodology IV,* Prentice Hall, (1978)

Hack M H T, *Analysis of Production Schemata by Petri Nets,* MIT Project, (1973)

Halliday M A, Vernon M K, *A Generalised Timed Petri Net Model for Performance Analysis,* International Conference on Timed Petri Nets, Torino, (1985)

Heijmink F et al, *Development of Tools for Designing OIS,* First European Conference on IT for Organisational Systems, EURINFO, (1988)

Hoare C A R, *Communicating Sequential Processes,* Communications of the ACM, (1978)

Holt A W , Commoner F, *Events & Conditions,* Applied Data Research Inc,(1970)

Huber P, Pinci V O, *A Formal, Executable Specification of the ISDN Basic Rate Interface,* 12th International Conference on Application and Theory of Petri Nets, (1991)

Hura G S, Singh H, *Some Design Aspects of Data Base Through Petri Net Modelling,* IEEE Transactions - Software Engineering, Vol SE-12, No. 4, (1986)

Jackson M A, *System Development,* Prentice Hall, (1983)

Jarayanta N, *Normative information model-based systems analysis and design, Journal of* Applied Systems Analysis, (1986)

Jelly I, Gorton I, Gray J, *Process Graphs: A Graphical Notation for Parallel Program Design,* PARLE '93 (1993)

Jensen K, *Coloured Petri Nets and the Invariant Method,* Theoretical Computer Science 14,(1981)

Jensen K, *How to find Invariants for Coloured Petri Nets,* Lecture Notes in Computer Science, 118, (1981)

Jensen K, *Computer Tools for Construction, Modification and Analysis of Petri Nets,* Lecture Notes in Computer Science, 225 (1986)

Jensen K, *Coloured Petri Nets: A Way to Describe and Analyse Real-World Systems - Without Drowning in Unnecessary Details*, 5th International Conference on Systems Engineering (1987)

Jensen K, Huber P, Shapiro R, *Hierarchies in Coloured Petri Nets*, 10th International Conference on Application and Theory of Petri Nets (1989)

Jensen K, *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use,* Unpublished text , (Publication 1993)

Jones C B, *Software Development: A Rigorous Approach*, Prentice Hall, (1980)

Kerridge J, *Dynamically Reconfigurable Arrays of Transputers to support Database Applications,* 7th occam User Group, Grenoble, France, (1987)

Kramer J, Ng K, Potts C, Whitehead K, *Tool Support for Requirements Analysi*s, Software Engineering Journal Vol. 3, No. 3, (1988)

Kung H T, Clementi E, Seitz C L,Steele G L, *Parallel architectures for supercomputing,* 11th World Computing Congress, (1989)

Lautenbach K, *Linear Algebraic Techniques for Place Transition Nets*, Lecture Notes in Computer Science, 254, (1987)

Lautenbach K, *Linear Algebraic Calculation of Deadlocks & Traps*, Advances in Petri Nets (1987)

Learmonth and Burchett Management Systems,*Automate Plus Manuals*, (1988)

Learmonth and Burchett Management Systems, *LSDM Part 1 and LSDM Part 2*, LBMS Consultants' Booklets, (1987)

Learmonth and Burchett Management Systems, *Systems Engineer*, LBMS Consultants Booklets (1990)

Loucopoulos P, Zicari R, *Conceptual Modeling, Databases and CASE,* John Wiley and Sons (1992)

Maggot J, *Performance Evaluation of Concurrent Systems Using Petri Nets,* Information Processing Letters, North Holland, (1984)

Martin J, *Strategic Data Planning Methodologies*, Savant Institute (1980)

Martinez J, Silva M, *A Simple and Fast Algorithm to Obtain all Invariants of a Generalised Petri Net*, International Conference on the Application and Theory of Petri Nets, (1982)

Meta Software, *CPN ML*, CPN Palette-Part 1(1988)

Meta Software, *The CPN editor*, CPN Palette-Part 3(1988)

Meta Software, *Hierarchies in Coloured Petri Nets*, CPN Palette - Part 2(1988)

Mumford E and Weir M, *Computer Systems in Work Design - The ETHICS Method,* Associated Business Press(1979)

Munro A, Dagless E L, *Real Time Control Including Concurrency, Part 2:Implementation,* Software Engineering Journal Vol. 1, No.4, (1982)

Munro A, Dagless E L, *Real Time Control Including Concurrency - Part 1:Design,* Software Engineering Journal. Vol. 1, No. 4, (1982)

Murakami K, Mori S, Fukada A, Sueyoshi T, Tomita S, *The reconfigurable parallel processor, Philosophy and architecture,* 11th World Computing Congress, (1989)

Nicol C, Crowe M K, Corr M E, Oram J W, Jenkins D G, IDEA - *An Incremental Development Environment for Ada,* Software Engineering Journal Vol 2, No. 6, (1987)

Olle T W et al, *Information System Methodologies: A Framework for Understanding,* Addison Wesley, (1988)

Olle T W, *An Information System Life Cycle and Related Concepts,* First European Conference on IT for Organisational System, EURINFO, (1988)

Pagnoni A, *Stochastic Nets and Performance Evaluation,* Proceedings of the Advanced Course on Petri Nets, (1986)

Perrot R H, Aliabadi A Z, *A Supercomputer Program Development System,* Software Practice and Experience, (1987)

Petri C A, *Kommunikatationen mit Automaten* (1962): Technical Report, Griffiss Air Force Base, New York [English Translation] (1966)

Petri C A, *Fundamentals of the Theory of the Asynchronous Information Flow,* IFIP Congress, North Holland,(1963)

Petri C A, *Concepts of Net Theory,* Slovak Academy of Science (1973)

Petri C A, *General Net Theory,* University of Newcastle (1977)

Petri C A, *Concurrency Theory,* Proceedings of the Advanced Course on Petri Nets (1986)

Post J, *Application of a Structured Methodology to Real Time Industrial Software Development,* Software Engineering Journal Vol. 1. No. 6 (1986)

Ramamoorthy C V, Ho G S, *Performance Evaluation of Asynchronous and Concurrent Systems using Petri Nets,* IEEE Transactions, Software Engineering Vol SE-6, No. 5 (1980)

Reisig W, *Petri Nets - An Introduction,* Springer Verlag (1984)

Roucairol G, *FIFO Nets,* Advanced Course on Petri Nets (1986)

Rozenberg G, *Behaviour of Elementary Net Systems*, Lecture Notes in Computer Science, 254 (1986)

Taubner D & Brauer W, *Petri Nets and CSP*, Proceedings of the 4th Hungarian Computer Science Conference (1986)

Thakker S, *Parallel Programming- Harnessing the Power,* IEEE Software, Vol. 6, No. 7, (1989)

Thiagarjan P S, *Condition Event Systems,* Aarhus University, Lecture Notes (1983)

Thiagarajan P S, *Elementary Net Systems*, Proceedings of the Advanced Course in Petri Nets (1986)

Tse T H, *A Unifying Framework For Structured Systems Development Models*, Private Communication, University of Kong (1986)

Varadharajan V, Baker K D, *Directed Graph Based Representation for Software System Design,* Software Engineering Journal, (1987)

Verrijn_Stuart A A, *Themes and Trends in Information Systems*, The Computer Journal, 30 (1987)

Voss K, *Using Predicate Transition Nets to Model and Analyse Distributed Database Systems*, IEEE Transactions on Software Engineering, 6, 6 (1980)

Voss K, *Stepwise Specification of a Distributed Database System*, 2nd. Conference on Distributed Computer Systems (1981)

Voss K, *Nets in Databases*, Lecture Notes in Computer Science, 255 (1987)

Voss K, *Nets in Office Automation,* Lecture Notes in Computer Science, 255 (1987)

Ward P T, Mellor S J, *Structured Development for Real Time Systems*, Yourdon Press, (1985)

Wilson B, *Systems: Concepts, Methodologies and Applications,* John Wiley and Sons (1984)

Wood-Harper A T, Fitzgerald G, *A Conceptual Framework based on features*, in Avison D E and Fitzgerald G (1988)

Wood-Harper A T, Episkopou D M, *Towards a Framework to choose appropriate information system approaches*, Computer Journal, 23, 3,(1986)

Wood-Harper A T, Fitzgerald G, *A Taxonomy of Current Approaches to Systems Analysis,* Computer Journal, 25,1 (1985)

Yourdon E & Constantine L L, *Structured Design*, Yourdon (1978)

Zuberek W M, *Timed Petri Nets and Preliminary Performance Evaluation*, Proceedings 7th Symp. on Computer Architectures, (1980)

APPENDIX A

The Program and Test Code for Duet Wright.

Program p111.

```
**********************************************
*    get consultant details
*
**********************************************
textmode()
select 1
use cons-a
select 2
use cons-b

answer = ' '
a new consultant? (Y/N) ' get answer
read
do while answer = 'y' .or. answer = 'Y'

  iperson = 0
  iname   = '        '
  igrade  = 0
  idiv    = 0
  ibill_rat = 0
  iaddress = '                              '

  @ 6,10 say 'consultant number' get iperson
  @ 7,10 say '             name' get iname
  @ 8,10 say '            grade' get igrade
  @ 9,10 say '         division' get idiv
  @10,10 say '     billing rate' get ibill_rat
  @11,10 say '          address' get iaddress
  read

  select 1
  go bottom
  append blank
  replace caperson with iperson
  replace caname with iname
  replace cagrade with igrade
  replace cadiv with idiv
  replace cabill_rat with ibill_rat
```

```
    replace caaddress with iaddress

    answer = ' '
    @ 13,10 say 'Enter a new skill? (Y/N) ' get answer
    read
    iskill = 0
    idacq  = 0
    do while answer = 'y' .or. answer = 'Y'
       @ 15,10 say 'skill code   ' get iskill
       @ 16,10 say 'date acquired' get idacq
       read

       select 2
       go bottom
       append blank
       replace cbskill with iskill
       replace cbperson with iperson
       replace cbdacq with idacq
       @ 13,10 say 'Enter a new skill? (Y/N) ' get answer
       read
       iskill = 0
       idacq  = 0
    enddo
    clear

    answer = ' '
    @ 4,10 say 'Enter a new consultant? (Y/N) ' get answer
    read
enddo
grphmode()
```

```
program p112
*
*     ADD  CONSULTANT DETAILS
*
textmode()
select 1
use cons-a
select 2
use cons-1

select 1
go top
do while .not. eof()

   sperson = caperson
   sname   = caname
   sgrade  = cagrade
   sdiv    = cadiv
   sbill_rat = cabill_rat
   saddress = caaddress

   select 2
   go bottom
   append blank
   replace c1person with sperson
   replace c1name with sname
   replace c1grade with sgrade
   replace c1div with sdiv
   replace c1bill_rat with sbill_rat
   replace c1address with saddress

   select 1
   delete
   skip

enddo
select 1
pack
grphmode()
```

program 1.1.3

```
textmode()
select 1
use cons-b
select 2
use acqsk-1

select 1
go top

do while .not. eof()

   sskill = cbskill
   sperson = cbperson
   sdacq  = cbdacq

   select 2
   go bottom
   append blank
   replace acs1skill with sskill
   replace acs1person with sperson
   replace acs1dacq with sdacq



   select 1
   delete
   skip

enddo
select 1
pack
grphmode()
```

program 2.1.1

```
textmode()
@ 2,2 say 'Making assignments'
count = 10
do while count  0
   count = count - 1
enddo

select 1
use rsk-1
select 2
use acqsk-1
select 3
use assig-a
select 4
use assig-b
select 5
use cons-1

select 1
go top
do while .not. eof()
   sclient = rs1client
   sproject = rs1project
   sskill = rs1skill
   scomm = rs1comm

   if scomm = '                    '
 @ 4,10 say 'Client '
 @ 4,20 say sclient
 @ 5,10 say 'Project'
 @ 5,20 say sproject
 @ 6,10 say 'Skill  '
 @ 6,20 say sskill
      assigned = 'n'
      select 2
 go top
 do while (.not. eof()) .and. (assigned = 'n')
    if acs1skill = sskill
 sperson = acs1person
 @ 8,10 say 'Consultant number, name has the required skill'
 @ 9,12 say sperson

 select 5
 go top
 locate for sperson = c1person

 if eof()
   sname = 'Not known'
 else
   sname = c1name
 endif
 @ 9,30 say sname

 answer = ' '
 @ 12,10 say 'Do you wish to make this assignment? (Y/N) ' get answer
 read

 if answer = 'y' .or. answer = 'Y'

    assigned = 'y'

select 1
replace rs1comm with 'assigned  '

select 3
go bottom
append blank
replace asaclient with sclient
replace asaproject with sproject
replace asaperson with sperson

select 4
go bottom
append blank
replace asbskill with sskill
replace asbclient with sclient
replace asbproject with sproject
replace asbperson with sperson
```

```
  endif
endif
        select 2
    skip
  enddo
  answer = ' '
  if assigned = 'n'
     @ 20,10 say 'No assignemt can be made ' get answer
       endif
  read
  @ 20,10 say '                    '
   endif

select 1
skip

enddo
grphmode()
```

program 2.1.2

```
textmode()
@ 2,2 say 'Entering assignments into the data base.'

select 1
use assig-a
select 2
use assig-1

select 1
go top
do while .not. eof()
   sclient = asaclient
   sproject = asaproject
   sperson = asaperson
   spesd = asapesd
   speh = asapeh
   srole = asarole

   select 2
   go bottom
   append blank
   replace asclient with sclient
   replace asproject with sproject
   replace asperson with sperson
   replace aspesd with spesd
   replace aspeh with speh
   replace asrole with srole

   select 1
   delete
   skip
enddo
select 1
pack
grphmode()
```

program 2.1.3

```
textmode()
@ 2,2 say 'Entering assigned skills into the database'

count = 1000
do while count  0
  count = count - 1
enddo

select 1
use assig-b
select 2
use assigs-1

select 1
go top
do while .not. eof()
   sclient = asbclient
   sproject = asbproject
   sskill = asbskill
   sperson = asbperson

   select 2
   go bottom
   append blank
   replace assclient with sclient
   replace assproject with sproject
   replace assskill with sskill
   replace assperson with sperson

   select 1
   delete
   skip
enddo
select 1
pack
grphmode()
```

program 311 enter project details

```
textmode()
select 1
use proj-a
select 2
use proj-b
select 3
use proj-c

Get Project Details'

answer = ' '
a new project? (Y/N) ' get answer
read
do while answer = 'y' .or. answer = 'Y'
  iclient = 0
  iproject = 0
  ipname = '        '
  iasd = 0
  ipeh = 0
  iphtd = 0

  @ 6,10 say 'Client number ' get iclient
  @ 7,10 say 'Project number' get iproject
  @ 8,10 say '        name' get ipname
  @ 9,10 say 'Act. St. Date ' get iasd
  @ 10,10 say 'Expected Hours' get ipeh
  read

  select 1
  go bottom
  append blank
  replace paclient with iclient
  replace paproject with iproject
  replace papname with ipname
  replace paasd with iasd
  replace papeh with ipeh
  replace paphtd with iphtd
```

```
   select 3

  go bottom
  append blank
  replace pcclient with iclient

  answer =''
  @ 12,10 say 'Enter a Required Skill? (Y/N) ' get answer
  read
  do while answer = 'y' .or. answer = 'Y'
     iskill = 0
 ishr = 0
 isrd = 0
 icomm = '              '

 @ 14,10 say 'Skill number    ' get iskill
 @ 15,10 say 'Skill Hrs. Req. ' get ishr
     @ 16,10 say 'Skill Req. Date ' get isrd
 @ 17,10 say 'Comments        ' get icomm
 read

 select 2
 go bottom
 append blank
 replace pbclient with iclient
 replace pbproject with iproject
 replace pbskill with iskill
 replace pbshr with ishr
 replace pbsrd with isrd
 replace pbcomm with icomm

 answer =''
     @ 12,10 say 'Enter a Required Skill? (Y/N) ' get answer
     read

  enddo
  answer = ''
  @ 5,10 say 'Enter a new project? (Y/N) ' get answer
  read
enddo
grphmode()
```

program 312 transfer project details to the database

```
textmode()
select 1
use proj-a
select 2
use proj-1

@ 2,2 say 'Transfer Project Details to the Database'

select 1
go top
do while .not. eof()
  sclient = paclient
  sproject = paproject
  spname = papname
  sasd = paasd
  speh = papeh
  sphtd = paphtd

  select 2
  go bottom
  append blank
  replace p1client with sclient
  replace p1project with sproject
  replace p1pname with spname
  replace p1asd with sasd
  replace p1peh with speh
  replace p1phtd with sphtd

  select 1
  delete
  skip

enddo
select 1
pack
grphmode()
```

program 3.1.3

```
textmode()
@ 2,2 say 'Entering Required Skills into the Data Base'
count = 1000
do while count  1
   count = count - 1
enddo

select 1
use proj-b
select 2
use rsk-1

select 1
go top
do while .not. eof()
  sclient = pbclient
  sproject = pbclient
  sskill = pbskill
  sshr = pbshr
  ssrd = pbsrd
  scomm = pbcomm

  select 2
  go bottom
  append blank
  replace rs1client with sclient
  replace rs1project with sproject
  replace rs1skill with sskill
  replace rs1shr with sshr
  replace rs1srd with ssrd
  replace rs1comm with scomm
  select 1
  delete
  skip
enddo
select 1
pack
grphmode()
```

program 3.1.4

```
textmode()
@ 2,2 say 'Marking clients active'
count = 1000
do while count  1
  count = count - 1
enddo

select 1
use proj-c
select 2
use client-1

select 1
go top
do while .not. eof()
  sclient = pcclient
  select 2
  go top
  locate for c1client = sclient
  if .not. eof()
    replace c1cs with 1
 select 1
 delete
  else
    answer = ' '
 @ 5,10 say 'Client number '
 @ 5,25 say sclient
 @ 5,30 say 'not set up'
 @ 7,10 say 'Press space bar to continue' get answer
 read
  endif
  select 1
  skip
enddo
select 1
pack
grphmode()
```

program 4.1

```
textmode()
@ 2,2 say 'Client Input'
select 1
use client-1
answer = ' '
u want to input another client (Y/N) ' get answer
read
do while answer = 'y' .or. answer = 'Y'

   iclient = 0
   icname = '        '
   icaddress = '                         '
   icont = '        '
   ics = 0
   icp = '        '
   @  7,10 say 'Clent number' get iclient
   @  8,10 say '      name' get icname
   @  9,10 say '    address' get icaddress
   @ 10,10 say 'Contact name' get icont
   @ 11,10 say ' Tel. number' get icp
   read

   select 1
   go bottom
   append blank
   replace c1client with iclient
   replace c1cname with icname
   replace c1caddress with icaddress
   replace c1cont with icont
   replace c1cs with ics
   replace c1cp with icp

   answer = ' '
   @ 5,10 say 'Do you want to input another client (Y/N) ' get answer
   read

enddo
grphmode()
```

program 7.1.1

```
textmode()
select 1
use timesl-b
select 2
use timesl-a




answer = ' '
a time sheet? (Y/N)' get answer
read
do while answer = 'y' .or. answer = 'Y'
   iperson = 0
   iclient = 0
   idate = 0
   iproject = 0
   ihtw = 0
   ihtg = 0
   iecd = 0

   @ 10,10 say 'Person number ' get iperson
   @ 11,10 say 'Weekend date  ' get idate
   read
   select 1
   go bottom
   append blank
   replace tlbperson with iperson
   replace tlbdate with idate

   answer = 'y'
   @ 13,10 say 'Enter a Time Sheet Line'
   do while answer = 'y' .or. answer = 'Y'

     @ 15,10 say 'Client number ' get iclient
     @ 16,10 say 'Project number' get iproject
     @ 17,10 say 'Hours this week' get ihtw
     @ 18,10 say 'Hours to go    ' get ihtg
```

```
    @ 19,10 say 'Exp. comp. date' get iecd
    read
  select 2
  go bottom
  append blank
  replace tlaperson with iperson
  replace tladate with idate
  replace tlaclient with iclient
  replace tlaproject with iproject
  replace tlahtw with ihtw
  replace tlahtg with ihtg
  replace tlaecd with iecd



    @ 20,10 say 'More time sheet lines?  (Y/N)' get answer
    read
    iclient = 0
    iproject = 0
    ihtw = 0
    ihtg = 0
    iecd = 0



   enddo
   answer = ''
   @ 5,10 say 'Enter a time sheet? (Y/N)' get answer
   read
enddo
grphmode()
```

Program true

```
trueflag()
```

test 1.1.2

```
use cons-a
go top
if eof()
 falsflag()
else
 trueflag()
endif
```

test 1.1.3

```
use cons-b
go top
if eof()
  falsflag()
else
  trueflag()
endif
```

test 2.1.1

```
falsflag()
use rsk-1
go top
do while .not. eof()
  if rs1comm = '        '
    trueflag()
  endif
  skip
enddo
```

test 2.1.2

```
use assig-a
go top
if eof()
   falsflag()
else
   trueflag()
endif
```

test 2.1.3

```
use assig-b
go top
if eof()
   falsflag()
else
   trueflag()
endif
```

test 3.1.2

```
use proj-a
go top
if eof()
   falsflag()
else
   trueflag()
endif
```

test 3.1.3

```
use proj-b
go top
if eof()
   falsflag()
else
   trueflag()
endif
```

test 3.1.4

```
use proj-c
go top
if eof()
   falsflag()
else
   trueflag()
endif
```

test 61

```
use timesl-2
go top
if eof()
  falsflag()
else
  trueflag()
endif
```

test 6.2.2

```
use bill-b
go top
if eof()
  falsflag()
else
  trueflag()
endif
```

test 6.3

```
use bill-c
go top
if eof()
  falsflag()
else
  trueflag()
endif
```

test 7.1.2

```
use timesl-b
go top
if eof()
 falsflag()
else
 trueflag()
endif
```

test 7.1.3

```
use timesl-a
go top
if eof()
 falsflag()
else
 trueflag()
endif
```

test 7.4

```
use timesl-1
go top
if eof()
 falsflag()
else
 trueflag()
endif
```