KNOWLEDGE REPRESENTATION USING PETRI NETS AND KNOWLEDGE TABLES

By

RAFAEL ORTIZ

Ingeniero Electricista

Universidad Nacional de Colombia

Bogota, Colombia

1986

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 1991

KNOWLEDGE REPRESENTATION USING PETRI NETS AND KNOWLEDGE TABLES

Thesis Approved:

Blayne E. Mayfield
Thesis Advisor

Huizhu Lu

Robert L Westerman

Thomas C. Collins
Dean of the Graduate College

ACKNOWLEDGMENTS

I want to thank to Dr. Blayne Mayfield, my principal advisor, for his advice, assistance and confidence. I also want to thank to the other members of my committee, Dr. Huizhu Lu and Dr. Robert Westerman for their suggestions and support during the development of this project.

I want to express my sincere thanks to the "Instituto Colombiano Agropecuario" for giving me the opportunity to further my education and professional qualities. I extend special thanks to Dr. Orlando Martinez, the head of the Department of Statistics, the understanding and support that I have received from him has been outstanding.

Sincere and special gratitude is extended to my wife, Maria Teresa, and my children, Natalia and Mariana for their patience, cares and encouragement during my studies; I could not have accomplished this important step without their support.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

During the last few years, the development of sufficiently precise notations for knowledge representation has been one of the main research topics of Artificial Intelligence and Computer Science [5]. One of the principal problems for computer scientists is to represent the real world for processing in computers, in other words, to make real-world knowledge suitable for processing by a machine [4], [5].

Knowledge representations are the methods used to encode and store facts, rules and relationships among objects or activities in a knowledge base [11], [24], [28].

Knowledge can be represented and structured in a variety of ways; the form of representation can significantly affect the efficiency with which knowledge can be stored and updated as well as the efficiency of inferencing.

The purpose of knowledge representation is to organize required information into such a form that people or computer programs can readily access it for making decisions, planning, recognizing objects, perform processes, drawing conclusions and performing other cognitive functions; thus, knowledge representation is an important topic for expert systems, machine vision, natural language processing, planning and control systems, and other systems.

Objectives

The aims of this project are:

a) to present a knowledge representation and reasoning method that combines Petri nets and a knowledge table [7] into a well-defined structure to support inference and reasoning for dynamic systems with concurrent activities;

b) to develop programs in C language that implement dynamic knowledge (event-driven reasoning), test them and evaluate their performance. With this software a user can analyze the behavior of a system and refine the model according to the results observed;

c) to analyze whether this model for knowledge representation can be used as a practical tool in the development of expert systems for asynchronous events or processes. The software will be tested with sequential as well as concurrent events or activities;

d) to analyze the algorithm for reasoning with complex or nested nets and give guidelines for its implementation.

Features of Using Petri Nets and Knowledge Tables as Knowledge Representations

Deng and Chang [7] introduce a new model for knowledge representation and reasoning based on Petri nets and a knowledge table; this model is called "G-net". Petri nets are introduced and extensively described by Peterson [19], and knowledge tables are presented by Chang and Ho [3].

The purpose of combining Petri nets and a knowledge table is to implement a new technique that facilitates the graph representation and the reasoning process of a system. A graph representation of a system is a model that expresses the constraints and relationships among activities or processes as a graph, so that reasoning algorithms can be implemented efficiently [5],[19].

This new model can be used to represent "static" and "dynamic" knowledge as described below. In this project, dynamic representation is implemented and its applicability to the desig of systems in which the activities are asynchronous and concurrent is analyzed.

Static representations are used to support reasoning about aspects of a system that are relatively constant, such as the interconnections between components. Dynamic representation can be used to investigate the time-varying characteristics of a system; dynamic representation may be viewed as a collection of procedures that, taken together, reflect the behavior of a system over time [16],[19],[28].

Dynamic representation is especially useful in situations where hypotheses about system behavior cannot be tested or verified using an actual system because that system is not available for test purposes or does not exist. When a real system has failed or instrumentation is not available, an inability to reason might occur. In these cases, use of the system in an experimental mode to test the consequences of the hypothetical situations would be very useful [28].

The main features of G-net model are summarized as follows:

- The reasoning algorithms are based upon Petri net theory and rules that have been studied from the birth of Petri nets.
- The G-net model can be implemented using the knowledge table representation, which is a convenient way of representing objects, activities and its relationships.
- The G-net model can support different types of activities throughout the net. Those activities can be simple or complex; the complexity of the activities might increase the time complexity of the reasoning process, but it does not affect the logic of the reasoning process.
- The G-net is a unified model that can represent dynamic knowledge and reasoning to carry out the subsequent logical activities in a process.

The G-net model in combination with a knowledge table is a flexible and modular technique because the representation is independent of the reasoning strategies; elementary G-nets can be combined to build complex nets, and G-nets can be applied to the specification of information systems and the simulation of asynchronous activities.

Outline of the Program's Development

In order to analyze the results of the algorithm for the reasoning process and to test it under different situations, a program was developed according to the following outline:

a) Define the data structures that will contain the data about places, transitions, and relationships (arcs) of the Petri net model. In addition, a data structure should be considered for the distribution of tokens in the Petri net.

b) Define the files to store the description of the system and the results of the reasoning process.

c) Write the code to read the input data that represents the system as Petri net and convert it to internal representation.

d) Write the code for checking the correctness of the model based on Petri net theory and the knowledge table.

e) Write the code for reasoning with dynamic nets (D-nets).

f) Write the code of the activities or external programs to test the model.

Tests for the Programs

This new technique will be tested with two kind of systems: one is a sequential reasoning system to determine a specific goal. The analysis for lime recommendations in acid soils was chosen for this purpose. The other test uses a system in which there are several concurrent activities. For this purpose a set of mathematical computations was selected to show concurrency with asynchronous activities.

To test the programs it was necessary to write some procedures and functions to simulate the activities and processes executed by the system during the reasoning process. In this stage, the results obtained with the tests are analyzed, and problems found during the development of the prototype of the model are identified.

CHAPTER II

OVERVIEW OF KNOWLEDGE REPRESENTATION TECHNIQUES

Representation of knowledge can be viewed as a technique in which knowledge is stored in a way that allows people or automatic systems to interpret or "understand" the relationships among elements of knowledge and to manipulate those relationships [28].

In the search for a precise and flexible knowledge representation many techniques have been developed; some commonly used techniques are procedural representation, rule-based systems, semantic networks, frames, Petri nets, blackboard and neural networks [24], [28]. In this study, the basics of these important techniques will be described.

The technique used in a specific knowledge representation should be selected carefully because each knowledge representation has relevance to particular types of knowledge; as stated by Walters and Nielsen [28] "none is applicable to all forms of knowledge". There is no set of rules or procedures that the designer of a system should follow; the designer should consider the advantages, disadvantages and limitations of every representation before selecting the technique to be used. For example, in selection, categorization or diagnostics, backward chaining may be the only mechanism needed [4]. In other cases, a straightforward procedural approach may be all that is needed for the reasoning process [11], [28].

A multiple-environment representation may be required in some cases, especially when the possible solutions can be enumerated [28].

Hunt [11] establishes that a blackboard representation can be effective in those situations where the reasoning process is complex and is to be controlled in a dynamic manner depending on partial results.

A D-net (dynamic Petri net, see Chapter III) is suitable to represent knowledge about concurrent activities [7]. It is modeled graphically and mathematically and presents several features for reasoning process.

Procedural Representation

Procedural representation is a method that represents knowledge about the world by a set of procedures, functions or programs that perform specific tasks [11], [28].

A procedure is a finite set of instructions for performing a task. It also can be defined as a program that executes an algorithm [11]; however, not all procedures embody algorithms; the reason is that an algorithm is a procedure that stops on all its inputs because the instructions are exhausted or a "stop" instruction is executed. In a procedural programming language, a procedure is a syntactic unit that can be parameterized in such a way that the same segment can be called from different places in the program, using different data or arguments each time.

Procedural representation of knowledge combines a number of items to form a solution. For example, XCON, which is a procedural system, configures DEC computer systems. From a customers order it decides what components must be added to produce a complete operational system and determines the spatial relationships among all of the components. XCON presents a set of diagrams about these spatial relationships to technicians who assemble the computer system [11].

Rule-based Representation

Many knowledge-based software development tools are designed on rule-based representation technology. Examples of rule-based systems are EMYCIN and OPS5, which are among the more comprehensive systems in which rules are conceptually represented as IF/THEN Statements [11], [28].

EMYCIN is a knowledge engineering language suitable for diagnosis and consultation type problems. It has user-querying facilities and is implemented in INTERLISP. OPS5 is a knowledge engineering language that supports generality in data representation and control structures; OPS5 has been implemented in MACLISP and FRANZLISP and is one of most widely used knowledge engineering languages.

Using IF/THEN statements or condition/action rules, knowledge can be accumulated into sets of rules. Figure 1 presents the structure of a typical rule-based system; the system is divided into a general-reasoning program (rule interpreter) and a file of rules, called the rule base, obtained from an expert [6], [11]. The reasoning program loads the rule base and use it to guide an interactive consultation with the user.

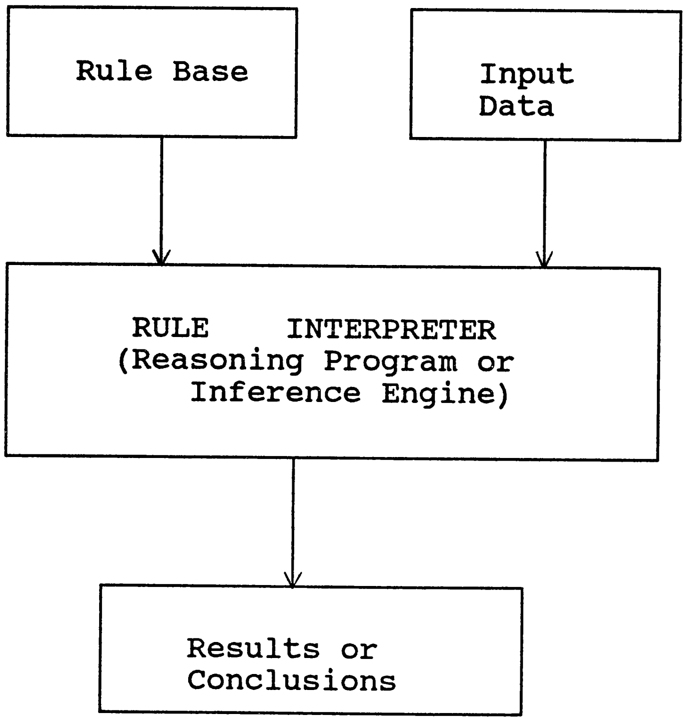

Figure 1. General Structure of a Rule-based System

The inference engine is the component of a rule-based system that controls its operation by selecting rules to use and determining when a solution has been found. An inference engine is also known as a control structure or rule interpreter [28]. The inference engine can process the rules in one of two ways: backward-chaining or forward-chaining.

In order to explain how backward and forward-chaining work it is convenient to define the terms predicate, antecedent, consequent and hypothesis. A predicate is a function that returns a true or false value, in other words, it is a statement about individuals in relation to themselves or other individuals; a predicate can be true or false when applied to an specific individual, so predicates are used to select among conditional alternatives [11], [26]. An antecedent is the left side of a production rule or the condition necessary to apply a procedure or a consequent. A consequent is the right side of a production rule or the result of applying a procedure. A hypothesis is a supposition or an unproved theory, also it is a consequent that do not appear as an antecedent in any other rule in a rule-set [11], [26].

The following example, taken from Walters and Nielsen [28], constitutes a rule base to illustrate how a rule-based system works.

```
Rule 1: IF   IT IS A WORKDAY
                  AND I AM IN THE OFFICE
          THEN  I EAT IN THE CAFETERIA

Rule 2: IF   I EAT IN THE CAFETERIA
          THEN  I EAT SOUP AND SANDWICH

Rule 3: IF   IT IS A WORKDAY
                  AND I AM OUT OF THE OFFICE
          THEN  I EAT OUT

Rule 4: IF   I EAT OUT
          THEN  I EAT CHINESE FOOD

Rule 5: IF   IT IS A WEEKEND DAY
                  AND I AM AT HOME
          THEN  I EAT AT HOME

Rule 6: IF   I EAT AT HOME
          THEN  I EAT PUMPKIN PIE

Rule 7: IF   IT IS A WEEKEND DAY
                  AND I AM OUT SHOPPING
          THEN  I EAT OUT
```

In this example the set of predicates is
{IT IS A WORKDAY, I AM IN THE OFFICE, I EAT IN THE CAFETERIA, I AM OUT OF THE OFFICE, I EAT OUT, IT IS A WEEKEND DAY, I AM AT HOME, I EAT AT HOME, I AM OUT SHOPPING};
the set of hypotheses is
{I EAT SOUP AND SANDWICH, I EAT CHINESE FOOD, I EAT PUMPKIN PIE}
and the set of consequents is
{I EAT IN THE CAFETERIA, I EAT SOUP AND SANDWICH, I EAT OUT, I EAT CHINESE FOOD, I EAT AT HOME, I EAT PUMPKIN PIE}.

The terminal predicates, which are the predicates that do not appear as consequents on any other rule in the rule base, are {IT IS A WORKDAY, IT IS A WEEKEND DAY, I AM IN THE OFFICE, I AM OUT OF THE OFFICE, I AM AT HOME, I AM OUT SHOPPING}.

In **backward chaining**, the inference engine identifies a set of one or more hypotheses and works backwards to locate known predicates that would provide support. The reasoning process begins with the inference engine taking the first hypothesis and locating all rules that have the hypothesis as a consequent. It then moves backwards from the consequent or goal to the premise of the selected rules and tests the truth of each predicate. If no predicate is determined to be true, then each unknown predicate is established as a new hypothesis and the process continues iteratively. This process forms a chain backward to the consequents of other rules, this is the reason for the term backward-chaining [11], [28].

If a hypothesis is selected from the above example, the inference engine operates as follows:

a. Select a hypothesis; for instance "I EAT SOUP AND SANDWICH"

b. Locate the hypothesis in the rule-set. Rule 2 contains the hypothesis as a consequent.

c. Examine the predicates of Rule 2 and determine whether they have been evaluated; if not, evaluate them. In the example, there is only one predicate to be evaluated (I EAT IN THE CAFETERIA); it evaluates to unknown. Now, the inference engine searches for another rule having the predicate of Rule 2 as consequent.

d. Locate a rule whose consequents contain the predicate "I EAT IN THE CAFETERIA"; Rule 1 contains that consequent.

e. Evaluate the predicates of this rule (Rule 1); the predicates (IT IS A WORKDAY, I AM IN THE OFFICE) are unknown.

f. Determine whether these are terminal predicates. In this case they are terminal predicates because they do not appear in any other rule.

g. Finally, ask the user for the values of unknown predicates; if they disprove the rule, select another path to the hypothesis. If no other path exists, try another hypothesis.

In backward-chaining the system keeps track of the values of the predicates and only asks that a value be supplied by the user when the predicate applies to a hypothesis being investigated and when the value of that predicate cannot otherwise be determined [4]. Backward-chaining is often used for selection applications, for instance, in diagnosing a particular problem.

Forward chaining is a problem solving technique which is characterized by working forward from known facts toward conclusions or goals. This technique starts with initial facts or knowledge, supplied by the user, and applies inference rules to generate new knowledge until one of the inferences satisfies a goal or no further inferences can be made [4], [11], [28].

Forward-chaining allows the user to infer implicit information from the existing information in a knowledge base; it is also useful to analyze changes when a new data item is added. A system that uses forward-chaining technique is called a data-driven system because it follows the conclusions obtained from the data or facts given [11],[28].

In forward-chaining no hypotheses are provided because the rules are not used to try to derive the truth of any particular consequent. Instead, they are used to derive all possible consequents from a set of predicates or facts that cause one or more predicates to evaluate to true.

Using the same set of rules in the above example, if the user provides the knowledge base with values such that the predicates "IT IS A WEEKEND DAY" and "I AM AT HOME" evaluate to true, then the forward-chaining inference engine operates as follows:

a. Locate the rules containing either of the provided predicates. Rule 5 and 7 are selected.

b. Select one of those rules. Rule 5 is selected, for instance.

c. Interpret (fire) the selected rule.
The predicates of Rule 5 are true, so the consequent is given the value true.

d. Now, a search is made for a rule containing "I EAT AT HOME" as a predicate. Rule 6 is found.

e. The rule (Rule 6) is interpreted and "I EAT PUMPKIN PIE" is given the value true.

f. Again, a search is made for a rule containing "I EAT PUMPKIN PIE" as a predicate. No such rule exists.

g. Rule 7 is evaluated to false because "I AM OUT SHOPPING" is false (or unknown).

h. The process terminates and the information obtained from the rule-based system is:
I EAT AT HOME and
I EAT PUMPKIN PIE.

This type of reasoning is appropriate for monitoring situations in which it is desirable to learn as much as possible about the state of the monitored system based upon the available data [4], [28].

Conceptually, the reasoning process must evaluate the predicate of each rule whenever a new fact is inserted into the knowledge base [28]; this is inefficient; therefore, many inference engines maintain an elaborated set of pointers, requiring only those rules containing new facts to be reevaluated.

In general, backward chaining goes from a conclusion to a set of premises to be evaluated, and forward chaining goes from a set of premises to a conclusion.

Rule-based representations have some disadvantages which can be summarized as follows:

a) Multiple evaluation of predicates. The inference engine often permits or requires the reevaluation of predicates more than one time, although a predicate is assumed to be evaluated only once [28]. If the inference engine being used permits or requires multiple evaluation of a predicate, then the designer must be very careful not to include terms within a predicate whose evaluation would have side effects such as contradictions, repetitive arithmetic operations or perhaps an infinite loop in the worst case.

b) Large sets of rules. The development of large sets of rules can pose two types of problems: inefficient execution and unmaintainable applications of expert systems [11], [28]. If the knowledge engineer does not find a way to decompose the problem into small independent rule-sets, then he/she should consider different representations of the

problem or different organizations of the knowledge base.
c) Uncertainty. Hunt [11] defines certainty as "the degree of confidence one has in a fact or relationship". It is very difficult to ensure that the desired relationships hold over a large set of rules to combine certainty factors. A certainty factor is a value or weight assigned to a relationship or event to specify the confidence level of that relationship or event. Although some tools assist the programmer in establishing certainties, the proper operation of a numeric certainty system across a large rule-set is a significant problem [28].

Frames

A frame can be defined as a collection of related information about a topic; this information can be factual or procedural (i.e., data or functions). A frame is basically a structure for holding various types of knowledge. Conceptually, a frame represents an item such as a physical object or a concept such as an idea; the contents of the frame then describe that item (e.g. characteristics, properties, behaviors). A frame can be viewed as a data record as used in programming languages as COBOL or PL/1; however, in this case the frame does not include functions, only data; in this sense, a frame consists of a set of named fields containing data that are in some way related; the relation depends on the particular problem or application

[11], [28].

Frames may be arbitrarily complex, and can have procedures and functions (pieces of code) attached to the slots to add or remove values from them. A slot is a feature, a component or an attribute associated with an object (or node) in a frame. A node is associated with an object, a concept or an event. The slots of a frame can contain default values which are helpful when frames are analyzed in the absence of full instantiation data (see figure 2) [26].

Rule-based and procedural representations can operate efficiently on frame-based representations. The object-oriented programming technique that is becoming popular has been developed on the frame foundation [26], [28].

Marvin Minsky conceived "frames" as complex data structures for representing stereotyped objects, events or situations. The idea was originated because many daily activities are instances of stereotyped situations such as going to work, shopping, driving a car, etc. A frame has slots for objects and relations that would be appropriate to these situations.

Figure 2 shows a frame-base representation of knowledge about the general configuration of a microcomputer. The features are associated with nodes representing concepts or

Frame: COMPUTER	
Slots	**Values**
Base Unit	16 MHz. Restriction: (value-type text) Restriction: (content-one-of: 16MHz, 20MHz, 25MHz)
Expansion Slots	3 Restriction: (value-type Integer)
Drive Bays	2 Restriction: (value-type Integer)
Video Adapter	CGA Restriction: (value-type Symbol) Restriction: (content-one-of None, CGA, EGA, VGA)
Monitor	Green Monochrome Restriction: (Procedure to select monitor depending on Video Adapter Restriction: (content-one-of None, Green Monochrome, Amber Monochrome Color, VGA b/w, VGA color)
Communication Board	None Restriction: (content-one-of None, 3270 Communications Adapter, Token Ring LAN Adapter)
Floppy Drive	1.2 Mb Restriction: (maximum-2-of 1.2 Mb, 1.44 Mb, 360 Kb, 740Kb)
Hard Drive	40 Mb Restriction: (content-one-of 40 Mb, 80 Mb, 100Mb, 200 Mb)
Keyboard	84-key Restriction: (content-one-of: None, 84-key, 101/102-key)

Figure 2. Frame Based Representation with Slots, Values and Restrictions

objects and they are described in terms of attributes (or slots) and their values. Each node's slots can be filled with values, according to the attributes of the object, to help describe the concept that the node represents.

Walters and Nielsen [28], and Charniak and McDermott [4] present complete introductions to frames with some examples.

An interesting characteristic of frames is the concept of "inheritance"; inheritance is a mechanism for passing knowledge or attributes from frame to frame down through a taxonomy of frames from general to specific.

The principal advantage of frame-based representation is that it provides a means for structuring a variety of types of data in the knowledge base. Other advantages are: it provides many characteristics of an object, through the concept of inheritance, once the type of that object has been identified, eliminating the need to derive these properties individually; it enables rules and procedures to be more generic; it enhances the maintainability of the knowledge bases.

Frames are useful in categorizing knowledge when that knowledge has some underlying structure. If the knowledge can be related to a set of objects or concepts, then at least a portion of the facts contained in the knowledge base can be clustered around those objects or concepts.

The main disadvantages are: as in object-oriented programming, the frame approach takes time and patience to master and use properly; one of the major dissatisfactions with frame reasoning is related with efficiency. The structures and capabilities of frames offer a variety of benefits, but they are achieved at a price: increased time complexity.

Semantic Networks

A semantic network describes the properties and relationships of objects, events, concepts, situations and actions by a directed graph consisting of nodes and labeled edges. Semantic networks formalize objects and values as nodes and connect nodes with arcs that indicate the relationships between them. Semantic networks are popular in artificial intelligence because of their naturalness to represent objects [4], [11].

Smith [26] defines Semantics as "the study of the meaning, intention or significance of a symbolic expression, as opposed to its form". Semantics is a constraint on a language understander because not all grammatically legal sentences have a meaning, for instance, "the stone was loud".

In semantic networks, concept types are organized in a hierarchy according to levels of generality, for instance (entity, animal, carnivore, lion) or (entity, thing, building, house). The relationships that hold for all concepts of a given type are inherited through the hierarchy by all subtypes [23]. For example, lion is a carnivore and also is an animal and an entity and inherits the characteristics of carnivore, animal and entity. In figure 3 the basics of a semantic network representing a car are shown; from this network it can be concluded that a "tire" is a "cylinder", a "car" has "tires", and an "engine" has "cylinders"; however, the cylinders for tires are different from the cylinders for engine, the differences are basically in type of material, dimensions and purposes.

For semantic networks it is necessary to define an associated vocabulary called "semantic primitives". This vocabulary is made up of "basic conceptual units in which concepts, ideas or events can be represented" [11]. Several attempts have been made to describe all primitives that are unique representations of entities or their attributes. A semantic primitive is defined by Smith [26] as "a primitive attribute of a domain that is used to build up facts in the data base".

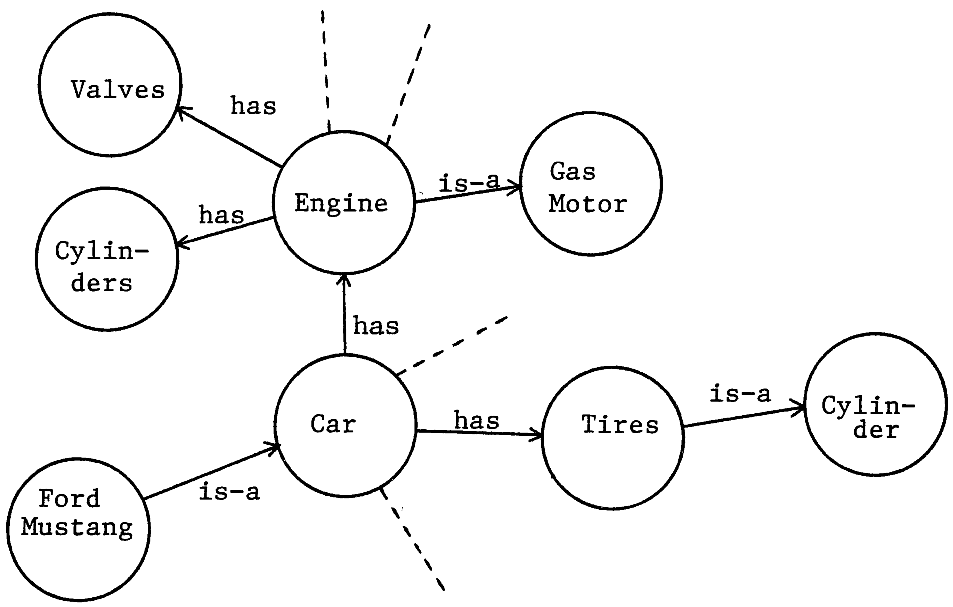

Figure 3. Semantic Network for Representing a Car

Functions and Boolean Arrays

Fordyce et al. [10] from IBM present an interesting model for representing knowledge using functions and Boolean arrays. The best known application of this technique is LMS - Logistics Management System - which is an advanced decision support system to dispatch, monitor and control the manufacturing flow of the IBM Burlington semiconductor facility.

The use of functions as a basic organizational unit of knowledge was originated in functional programming languages such as APL2 and LISP and their mathematical concept of functions.

In the technique presented by Fordyce et al. [10] there are two basic aspects: 1) knowledge is viewed as a functional mapping between input and output variables, where the functions are expressed as fact tables or bases and procedural modules; 2) the function network can be represented with boolean arrays. The use of tables to store and represent knowledge has its origins in general array theory, relational data bases, APL2 and PROLOG; the use of Boolean arrays and operations to efficiently handle logical processing is also well established.

Tables can represent a functional relationship between input and output variables; small procedures can be defined to describe functional relationships that can carry out computations on the input variables to generate the output variables; the linkages between tables and functions represent composite functions, for instance,

$$W = f(X,Y) = X^2 + 4Y$$
$$Z = g(W,V) = 3W + 2V$$

is a system of 2 equations; this system is expressing a functional mapping of the input variables V, X and Y to the output variables W and Z. Z has a dependency on X and Y through the variable W, this is called a composite function, which is represented as follows:

$$Z = h(X,Y,V) = g[f(X,Y),V]$$

Using Boolean arrays and Boolean operations it is possible to determine automatically how the functions of a system are related to one another and generate the network of functions or tables that represents the system. In the Boolean matrix a cell is assigned a 1 if the variable is in the "input portion" of a function; if the variable is not in the input of a function, the corresponding cell is assigned a zero.

All the concepts on this technique have been applied using APL2 programs because this language is based on matrix operations [10]; this technique has been effective in real-time applications, transaction-based and knowledge-based systems.

Petri Nets

A Petri net is an abstract and formal model to represent and analyze information flow [19].

The Petri net has increased in acceptance as a flexible and powerful model of systems of asynchronous and concurrent computation. Actually, the properties, concepts and techniques on Petri nets have been developed in a search for more natural, simple and convenient methods for representing and analyzing the flow of information and control in systems, especially when the systems may exhibit asynchronous and concurrent activities [16]. So far, the most important use of Petri nets has been the modeling of systems of events that can occur concurrently.

A generalized Petri net is a five-tuple defined by:

$$\text{Petri net} = (P, T, I, O, \mu)$$

where:

$P = \{p_1, p_2, \ldots.p_n\}$ is a finite set of places. A place can be an static object or a dynamic activity.

$T = \{t_1, t_2, \ldots..t_m\}$ is a finite set of transitions. A transition represents the functional relationship between dynamic objects or the semantic relationship between static objects.

$P \cap T = \emptyset$

$I: P^{\infty} \rightarrow T$	is an input function, a mapping from places to transitions.
$O: T \rightarrow P^{\infty}$	is an output function, a mapping from transitions to places.
$\mu: P \rightarrow N$	is a marking, a mapping from places to non-negative integers N.

A Petri net is represented by a bipartite directed multigraph containing two types of nodes: places and transitions. Figure 4 shows a graph representation of a simple Petri net; the places are represented by circles and the transitions are represented by bars.

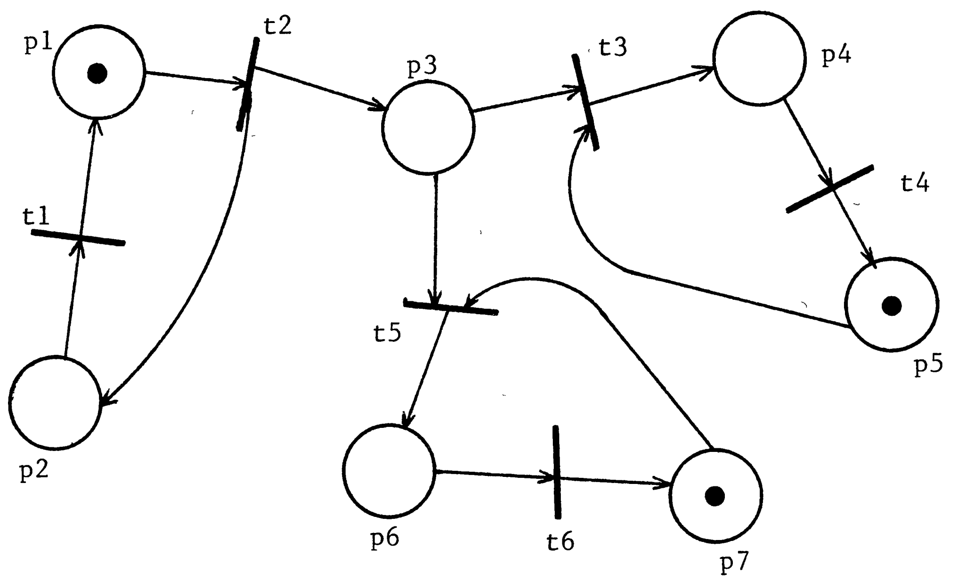

Figure 4. A Graph Representation of a Petri Net

These nodes, places and transitions are connected by directed arcs from places to transitions and from transitions to places. The formal definition of the Petri-net of figure 4 is:

Petri net = (P, T, I, O, μ)

where :

$P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$

$T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$

$I(t_1) = \{p_2\}$	$O(t_1) = \{p_1\}$
$I(t_2) = \{p_1\}$	$O(t_2) = \{p_2, p_3\}$
$I(t_3) = \{p_3, p_5\}$	$O(t_3) = \{p_4\}$
$I(t_4) = \{p_4\}$	$O(t_4) = \{p_5\}$
$I(t_5) = \{p_3, p_7\}$	$O(t_5) = \{p_6\}$
$I(t_6) = \{p_6\}$	$O(t_6) = \{p_7\}$

The marking vector, which represents the number of tokens in every place, is $\mu = (1,0,0,0,1,0,1)$

"Token" is a primitive concept for Petri nets; a token is a "message" that can be transmitted between places [16], [19]. This message can contain simple or complex information about the net; on a Petri net graph, tokens are represented by dots inside places as is shown in figure 4. A Petri net with tokens is called a marked Petri net; tokens are moved by the execution (firing) of the transitions of the net, so that the marking may change as a result of the firing of any transition.

In many sciences a phenomenon is studied by examining not the actual phenomenon itself but rather a model based on mathematical terms. By the manipulation of the representation, it is hoped that new knowledge about the phenomenon, and the model itself, will be obtained without the cost and inconvenience of manipulating the real phenomenon. Petri nets are a modeling tool. They can model systems and especially events, conditions, and the relationships among them; the occurrence of events may change the state of the system, causing some of the previous conditions to cease holding, and causing other conditions to begin to hold [16].

In the Petri net model, events or activities of places and transitions can occur independently [7], [19]. Thus, there is no need to synchronize the actions of events; however, when synchronization is required, the situation is also easily modeled [16]. Therefore Petri nets are ideal for modeling systems of distributed control with multiple activities occurring concurrently [13], [19], [20].

One of the most important features of Petri nets is their asynchronous nature. There is no inherent measure of time; the only important property of time, from a logical point of view, is in defining a particular ordering of the occurrence of events. A Petri net must contain all necessary information to define the possible sequences of events of a modeled system.

A Petri net is non-deterministic. If at any time more than one transition is enabled, then any of the several enabled transitions may fire; this feature of Petri nets reflects the fact that in real life situations several activities may happen concurrently. The order of occurrence of events is not unique (perhaps random), so that any of a set of sequences may occur.

Nondeterminism is advantageous from the modeling point of view, but it introduces more complexity in the analysis of Petri nets. This complexity can be reduced if the firing of a transition (execution of an event) is considered to be instantaneous (zero time).

Another important feature of Petri nets is their ability to model systems hierarchically. This means that a net may be replaced by a single place or transition for modeling in a higher level of abstraction. On the other hand, a place or transition may be replaced by subnets to provide more details in the model refinement [7], [8].

Neural Networks

The goal of neural networks research is to understand complex human performance, such as how people learn to play the piano. Even the simplest models provide insights into how the learning process occurs [17], [21].

When human beings learn something, there are several activities involved such as remembering, understanding, storing and retrieving; but brain surgeons say that there is more: firing neurons, making new connections and retraining behavior patterns.

There has been a need for a way to solve problems that cannot be handled efficiently by digital means. Neural networks technology is an attempt to simulate the behavior of the brain from a physiological view, especially in that kind of problems that cannot be efficiently handled by digital means [17], [21].

A biological neuron consist of a cell body, axons and dendrites; the cell body is the nucleus or "main processor" of a neuron, which processes the information it receives; an axon is that part of a nerve cell through which impulses travel away from the cell body, a dendrite is the branched part of the cell that carries impulses toward the cell body. The points of contact between adjacent neurons where nerve impulses are transmitted from one to the other are called synapses. An artificial neuron or "unit" emulates the axons and dendrites with connections or arcs and the synapses by simulating electric resistors with weighted values.

Neural nets are computer models inspired by the brain. A neural net consists of processors or units that simulate the behavior and properties of neurons. Each of these units or "neurons" receives inputs that can be excitatory and inhibitory, from other units; if the strength of the signal exceeds a given bound, the unit sends signals to other units. Each of the many connections or synapses among units has its own strength, or weight (like a multiplier) that can be adjusted as the net performs new tasks or operations (see figure 5).

Perhaps the most interesting aspect of neural nets is its capability of "learning". Instead of programming a neural net, you "teach" it to give acceptable answers. As stated by Caudill [2], the knowledge in a neural net is stored "in the pattern of weights and connections in the network"; this means a user can input known information, assign initial weights to the connections within the

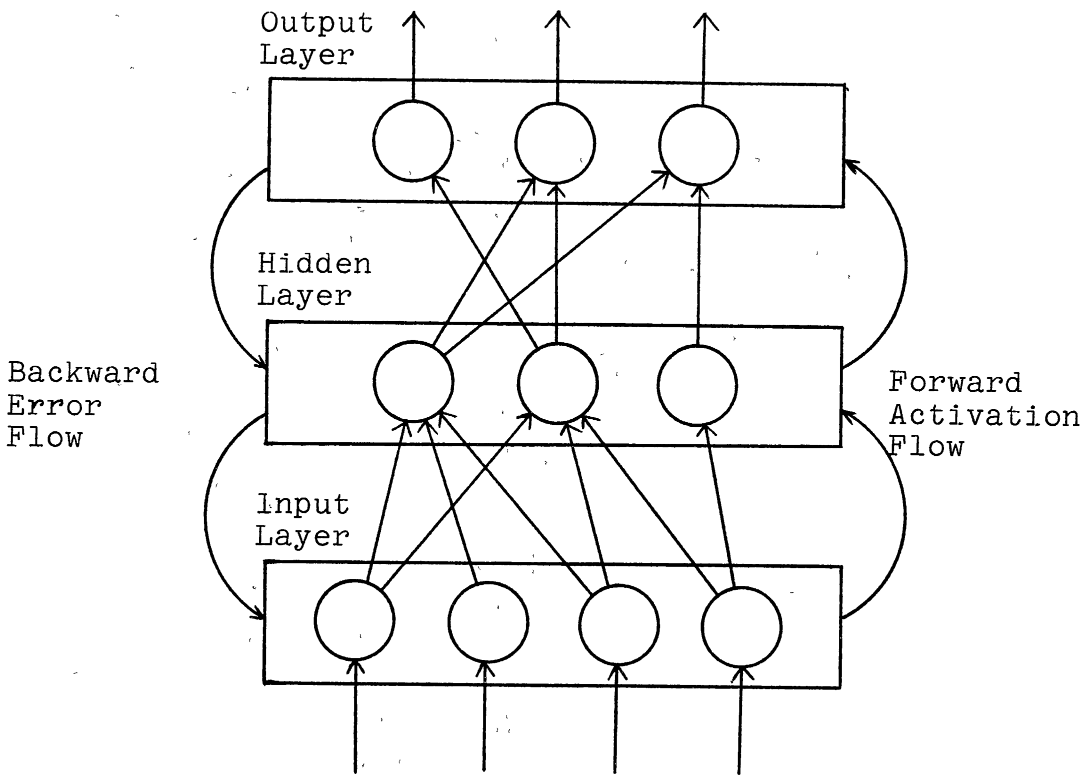

Figure 5. Neural Network with a Hidden Layer

architecture, and run the network over and over until the output is satisfactorily accurate, the net can adjust the weights by using several criteria. The net contains a weighted matrix of interconnections that allows to learn and remember [2], [17], [21].

When they work correctly, neural nets provide some important benefits, such as the ability to take incomplete data and produce approximate results. They are fault-tolerant because of their parallelism, speed and trainability [27].

However, because neural nets simulate the brain, they do not handle numbers well, especially for accurate answers. Accuracy, computational power and logic are not strong aspects of neural nets [2]; another weak aspect is that they cannot explain how they solve a problem. The difficulty with this aspect is that in this stage of technology the man does not know completely how the brain works. What is available now are artificial neural networks that run on digital machines to develop general principles to explain human information processing [21].

Artificial neural nets are being used for a variety of applications like financial analysis, database management, medical diagnosis, fuzzy or incomplete information and some kind of process control [2].

Neural network models consist of processing elements, interconnection topologies, and learning schemes [27]. Processing elements interact each other depending on how they are interconnected; when a neural net is setting up, a variety of criteria is used to define specific interconnections and determine its architecture.

Obermeir and Barron [17] say that neural network "memory" is measured by the number of interconnections as the memory in a digital computer is measured in bytes; in the same way the neural net's speed is measured in interconnections per second.

Each processing element or neuron has a number of inputs, a small set of possible states, and an output that is a function of the inputs; each input to the neuron has a weight value that usually is between -1 and 1.

Training a neural network is a matter of adjusting weights, either manually or automatically. Obermeir [17] establishes that the learning process of a neural net is possible in one of three ways: supervised, unsupervised or self-supervised.

Supervised learning occurs when trial-and-error inputs are provided that teach the network correct and incorrect responses. **Unsupervised** learning consists of entering and adjusting data without human intervention. **Self-supervised** learning occurs when the net monitors itself and corrects errors in the interpretation of data. This can be done by feedback through the network.

One important characteristic of neural nets is "stability"; after the initial weights are set, the user enter data into the network. This process causes the net to pass through state changes and finally reach stability. A net achieves stability when the weight values associated with the "units" stop changing [2], [27].

A neural network layer is a set of units or "neurons" that are at the same level in the network (see figure 5). Initially, neural nets consisted of only one or two layers to represent knowledge, but adding more layers allows the system to form a better representation of the problem [17], [27]. Today neural nets are composed of several layers such as the model presented in figure 5, this hierarchical approach is more powerful because neural networks can generate their own internal representation in the so-called hidden layers.

Touretzky and Pomerleau [27] present the characteristics and features of the layers in a hierarchical net. A hierarchical network consists of an input layer, an output layer and one or more hidden layers. A hidden layer is a set of units that are not directly connected to the input or output layers. Reducing the size of a hidden layer not only increases the rate of the simulation but also improves the network's performance. A network with too many hidden layers can simply memorize the correct response to each pattern in its training set instead of learning a general solution.

Neural networks can learn using an algorithm called back-propagation [27]. With back-propagation an expert provides the network with samples of inputs and desired outputs, over and over, until the network learns by adjusting its weights. If the net solves the problem, it will have found a set of weights that produces the correct output for the given input. Whatever knowledge the network acquires is encoded in its numerical weights. Unlike expert systems, neural networks do not automatically explain their reasoning.

Back-propagation consists of two passes. In the forward pass, inputs proceed through the network and produce a certain output; then, in the backward pass, the difference between the desired output and the actual output generates an error signal which is propagated back through the network to teach it to come closer to producing the desired output (see figure 5).

Obermier and Barron [17] say: "Neural nets won't replace data-base and knowledge-based processing because they do not work well with numbers or cut-and-dried information". Maybe in the next few years the first practical neuron-like circuit will appear; in this event a neural network could be used as a coprocessor controlled by a host digital computer.

Blackboard Representation

The blackboard approach is a system architecture that uses a data base or records that are accessible to several processes called knowledge sources [28]; each process can "write" on the blackboard, and all the other processes can "read" what has been written, and respond in a similar manner. This approach took that name because the system organizes and processes knowledge in a fashion analogous to a group of people working around a blackboard; each person represents a specialized source of knowledge about some aspect of the problem. A leader provides the control function, guiding and coordinating the activities of the knowledge sources as well as sequencing their access to the blackboard [11].

Actually, a blackboard system is not a particular form of knowledge representation as it is a way of organizing and processing knowledge represented in other forms. A blackboard system can be seen as a framework in which knowledge can be arranged in such a way that it can be distributed and shared among a number of cooperating processes; the knowledge about a particular problem can be distributed to a set of specialists or knowledge sources, each of which has a particular area of expertise. The shared portion of knowledge is encoded on the blackboard through which the specialists communicate; the distributed part resides with the individual specialists who operate independently of each other. The communication among them is through the blackboard [28].

A blackboard structure provides room for many different solution approaches, so it can be viewed more as a philosophy or a set of guidelines than as a very detailed process for knowledge representation and reasoning.

The project HEARSAY II for speech recognition was developed with the blackboard concept at Carnegie-Melon University [28]. The blackboard approach has been applied in a variety of fields such as real-time data processing, speech recognition and signal processing, and scheduling and planning. A blackboard representation might be considered for problems that naturally decompose into a number of smaller and independent structures.

In figure 6 a schematic of the blackboard system is presented; there are 3 basic components: knowledge sources or expertise (KS_1,..KS_n), blackboard (knowledge storage and communication), and control, which is the problem solving strategy.

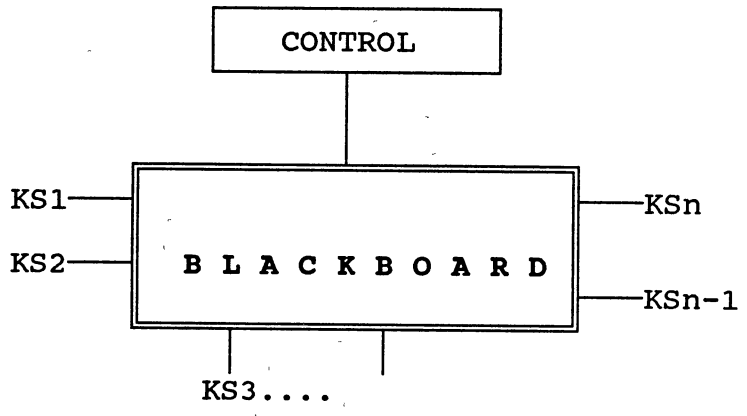

Figure 6. General Schematic of a Blackboard System

The blackboard represents the communication medium through which the specialists or knowledge sources communicate their conclusions, findings, or requests for data to each other; thus, the blackboard is the source of all data on which a knowledge source operates and the destination for all conclusions from a knowledge source [11], [28].

The blackboard contains two kinds of knowledge: static and dynamic. Static knowledge consists of factual data that refer to initial conditions, parameters, and relationships;

dynamic knowledge is the knowledge that is generated during the execution of the application. New facts, communications, hypothesis, goals and suggestions are considered dynamic knowledge.

The blackboard structure probably offers the best means of representing procedural knowledge [28]; thus, if there exists difficulty in converting domain expertise to a non-procedural form, that knowledge might be represented procedurally in a set of knowledge sources as is shown in figure 6.

The major advantage of a blackboard system (viewed as the application of procedural knowledge in a much more structured way) is also, however, its principal weakness. By training and by experience many knowledge engineers or applications developers have computing backgrounds that emphasize procedural programming, and they tend to think in procedural terms. This aspect causes a common problem in rule-based applications: the incursion of procedural knowledge in what should be a non-procedural representation. A developer without prior experience in non-procedural thinking and non-procedural knowledge representations can slip easily back into using procedural terms, because the blackboard model easily accommodates such thinking. In conclusion, the blackboard entices the novice to think in procedural terms rather than representing aspects of the domain knowledge in some non-procedural form.

Knowledge Table

The knowledge table representation is a structure proposed by Chang and Ho [3]. A knowledge table is composed of knowledge objects that can be values, activities, expressions, concepts, or other knowledge tables. The knowledge objects can be static or dynamic, and different semantic or functional relationships among those objects can be defined.

A value can be a symbol or number, an activity can be a single action or a compound action, an expression is an arithmetic or Boolean expression, and a concept or entity refers to anything that contains various aspects or properties. The use of tables to store and represent knowledge has origins in general array theory, relational data bases, APL2 and Prolog [10].

A detailed description of the knowledge table representation is given in Chapter 3, in combination with Petri nets.

CHAPTER III

KNOWLEDGE REPRESENTATION AND REASONING USING PETRI NET AND KNOWLEDGE TABLE

Introduction

Knowledge can be classified into two basic types: fact knowledge (F-type knowledge), which refers to what has been explicitly specified to be true, and inference knowledge (I-type knowledge), which specifies the cause-effect relationship among the objects, from which new fact knowledge can be derived [7].

F-type knowledge is represented by objects or facts and I-type knowledge is denoted by inference rules for sequential events. The real world can be modeled by a collection of knowledge objects or facts and the cause-effect relationship among them; however, F-type knowledge and I-type knowledge are sometimes insufficient to efficiently support inference and reasoning, especially in concurrent or parallel activities like those presented in the industry or in a computer system.

Knowledge can be represented in static and dynamic models. A static representation can be viewed as a collection of initial conditions, parameters, facts and relationships; examples of such representation are a semantic network, an electronics diagram and a hydraulic diagram. A dynamic representation of a system may be viewed as a collection of nodes and arcs. In this case the analyst's concern is about the effects of changing inputs over time and the propagation or effects of those changes in the system [28].

Walter and Nielsen [28] say "A static model is analogous to a set of production rules". A set of data is provided as input to the model or rule-set and, as a result, another set of data is returned. This mode of model usage is analogous to forward chaining or backward chaining with a set of production rules.

Static representations can be used to support reasoning about aspects of a system that are relatively constant, such as the interconnections between components. Static representations are used with diagnostic or failure isolation applications for example in hydraulic and electric systems; the benefits of static representations are clarity, familiarity, easy maintenance, simplification, efficiency and easy application.

The dynamic representation of a system can be used to investigate the time-varying characteristics of a system; the analyst's concern is about the system's dynamic behavior. The dynamic representation may be viewed, in contrast to the static one, more like a collection of procedures that, taken together reflect the behavior of the system through time. One of the main disadvantages of dynamic models for knowledge representation is that the development and representation of the appropriate set of basic principles is a nontrivial task and sometimes might be of unexpected dimensions [16], [19].

The new model based on Petri nets and knowledge table representation can have static and dynamic objects. It is static when the objects are grouped according to their semantic relationships such as semantic networks and frame structures. The model is dynamic when it groups dynamic objects according to their functional relationships such as Petri nets [19], [5].

Usually these two models are not compatible; that is, a semantic network is not capable of modeling the functional relationships among dynamic objects, and a Petri net is not suitable for representing semantic relationships among objects.

The G-net is a unified model that can represent both static and dynamic knowledge and also support inference and reasoning. There are four basic reasoning algorithms in the G-net model [7]:

1. Inheritance Reasoning. Inheritance reasoning is the form of reasoning to infer properties of an object based on the properties of its ancestors. Inheritance also may be defined as the process of determining properties of an object by looking up properties attached to objects that are above it in the conceptual hierarchy. The results of this process are all the properties of an object.

2. Recognition Reasoning. The recognition reasoning is in the opposite of inheritance; recognition is the process of finding an object or concept that best matches a given

description consisting of a set of properties. The results of recognition are all the objects which exhibit the given properties.

3. Event-driven Reasoning. This reasoning process is called event-driven because it is based on the occurrences of events. The goal of event-driven reasoning (reasoning for dynamic nets) is to determine the subsequent activities based on current events.

4. Complex Dynamic Reasoning. This algorithm is a higher level reasoning based on the G-net model in order to control the flow of the reasoning process when every object in the model is another G-net.

The first two algorithms are the bases for some Artificial Intelligence procedures, and they have been studied extensively in different ways using techniques such as semantic networks, predicate calculus, rule-based systems, breadth-first search, depth-first search, etc. [4], [24].

The algorithms for dynamic knowledge are useful for analyzing concurrent and sequential activities. The major use of Petri nets has been the modeling of systems of events that occur concurrently; this model is based on the concepts of asynchronous and concurrent operations by the parts of a system [19].

According to Deng [9], the Department of Computer Science of the University of Pittsburgh is developing an editor for the knowledge table representation, and they plan to develop programs in Prolog for reasoning.

Knowledge Table and Petri Net Model

G-type knowledge model is a combination of knowledge tables and Petri nets [7], [3], [19].

Knowledge Table

A knowledge table is a non-empty set of knowledge slices. A knowledge slice of a knowledge table is a nonempty set of knowledge objects. A knowledge object can be a value, an activity, an expression, a concept or a structure composed of other knowledge objects. The slice may contain knowledge objects of different types; however the basic types of slices are:

a. F-type, for objects of any type (except expressions)

b. I-type, which consists of at least two objects (exp,cons); exp is of type expression and cons is any type of object except expression. Cons is evaluated if the expression exp is evaluated "true".

c. G-type, which is used to represent a set of static knowledge objects or a set of interrelated concurrent and/or sequential activities.

The type of knowledge table depends on the relationship among the slices; therefore the knowledge table can be I-type when all the slices are of I-type, F-type when all the slices are of F-type, G-type when all the slices are G-type. There is another type of knowledge table, the control table, which is used in the analysis of complex G-nets. A complex G-net represents a mixed type of knowledge hierarchy. A control table is used when a G-net contains a mix of knowledge types.

The knowledge table has several features [3] :

a. A knowledge table can contain other knowledge tables. This facilitates the construction of complex knowledge structures.

b. A knowledge table can be partially defined at first and incrementally refined later on.

c. A knowledge table system can represent other commonly used knowledge representation models, such as rule-based, semantic nets and frame structures.

The main idea behind the knowledge table is to group together in one structure knowledge slices to make processing more efficient.

<u>Petri Net and G-net</u>

Deng and Chang [7] introduce the G-Net model, which is based on Petri Nets. In the G-net representation the knowledge objects (static or dynamic) are modeled by the places of a Petri net and the relationships among the objects are represented by the transitions of the Petri net.

The G-net model is a combination of F-type and I-type representations to facilitate the representation of a set of well structured activities and for concurrent activities, or to represent control knowledge. For concurrent activities F-type and I-type represented independently are not adequate. A G-net can be used to represent a set of static knowledge objects and their semantic relationships, as well as a set of concurrent and/or sequential activities. In the present work the attention is concentrated on the latter representation in which the functional relationships among a set of dynamic activities can be specified.

Formally, a G-net is defined as a seven-tuple that is an extension of the definition of a Petri net:

G-net = (k, P, T, u, I, O, D)

where: k represents "static" or "dynamic"; if k="static", the net is used to represent static knowledge and semantic relationships; if k="dynamic" the net is used to represent dynamic knowledge and their functional relationships.

D indicates properties of the semantic relationships when k="static",

P, T, u, I and O have the same meaning as in Petri nets;

however, this definition takes in account static and dynamic knowledge (k parameter), and the interest of this project is only in dynamic and concurrent activities; in order to simplify the terminology the model can be reduced to:

G-net = **(P, T, u, I, O)** = **D-net** = Dynamic net

This reduction can be done without loss of generality because the parameter k is used to indicate whether the G-net is "static" or "dynamic", and the parameter D is used to indicate the properties of the semantic relationships in the G-net [7], and it is defined only when k="static".

Now, the net is called a D-net (Dynamic net) which basically is a Petri net. A D-net is a five-tuple as established above, where:

$P = \{p_1, p_2, \cdots, p_n\}$

P is a set of places which is finite and is used to represent a "dynamic" set of objects.

$T = \{t_1, t_2, \ldots, t_m\}$
T is a set of transitions which is finite and is used to represent the functional relationships among dynamic knowledge objects. A logical predicate 'w_i' is associated with each t_i to define the functional relationship among a subset of places connected by t_i. The predicate constitutes a necessary condition for t_i to be firable.

$u : P \rightarrow 2^{N\times\{b,w\}}$
u is a marking function which determines the initial token distribution; N is the set of natural numbers. The "tokens" is a primitive concept for Petri nets; they reside in the places of the net and the number and position of tokens can change during execution. The tokens in a D-net can have the colors: black(b) and white(w). Black and white tokens are used in dynamic reasoning; the use of this tokens (b,w) will be described later in this chapter.

$I: T \rightarrow 2^{P\times N\times\{b,w\}}$
I is an input function, where N is the set of natural numbers. I is an input function mapping from transitions to the power set of $\{P\times N\times\{b,w\}\}$. The input function and the output function together define the relationship among places and transitions.

$O: T \rightarrow 2^{P\times N\times\{b,w\}}$
O is an output function. In Chapter V there is an example on how to define the input and output set of functions for every transition in the D-net representation.

As in Petri-nets, the graphic representation of D-net denotes places as circles and transitions as bars and the input and output functions as directed links (arrows) connecting the places and transitions.

Associated with each transition T_i there is a logical predicate w_i which constitutes a necessary condition to make a transition firable or executable. A transition is active if the logical predicate is true; otherwise it is passive.

In Petri nets, information can be attached to each token as a "token color" and each transition can be enabled in several ways depending on the conventions established by the execution algorithm about the occurrence of different "token colors" [13].

For the reasoning process on a D-net it is necessary to include two token colors: white(w) and black(b). A transition is enabled if there is at least one white token on each of its input places. A transition is firable if and only if the transition is active and enabled. After firing a transition, one or more black or white tokens will be added to every output place of T_i according to the output function for that transition.

A place is active if it is allowed to perform its activity, i.e., if it holds at least one white token. A place is passive if it is prohibited to carry out the activity in the place, i.e., if it does not have any white token or receives one or more black tokens. Therefore, a white token in a place p_i means that the activity in p_i is enabled; a black token in a place p_i means the activity in p_i is forbidden. If a black token is added to a place the effect is to reset the number of white tokens to zero in that specific place, so that black tokens are not stored.

The marking μ represents the current state of the D-net; therefore, the combination of marking μ and transitions T will determine the invocation of subsequent activities in the reasoning process of the system represented by a D-net.

Formal Specification of D-net Model and Knowledge Table

Since the D-net model is basically a graph model [19], the knowledge table can be used to implement the D-net model according to the following assertions: the knowledge table used to store D-type knowledge is called a D-type knowledge table, and the knowledge slices in a D-type table are D-type knowledge slices (the places or objects are "dynamic" activities), which are used to represent places and transitions in D-net.

The knowledge slices in a knowledge table can be divided in four subgroups according to the definition of Petri nets:

Place-type:	It is used to store the information about a place in the D-net.
Trans-type:	It is used to store the information about a transition in the D-net.
In-type:	Describes the input function of a transition in a D-net.
Out-type:	Describes the output function of a transition in a D-net.

Place-type is defined as:

$(p, altp1(ti_1, ti_2 ...., ti_p), altp2(to_1, to_2 ... to_q))$

where:
p is an activity or process.
altp1, altp2 are two flags or alerters whose function is to monitor the event which occured and the movement of tokens in the corresponding place p.

($ti_1 \ldots ti_p$) are the transitions which have place p as one of the input places.
($to_1 \ldots to_q$) are the transitions which have place p as one of the output places.

Trans-type is defined as:

(t, altt,(pi_1,pi_2....,pi_m),(po_1,po_2...po_l)),

where:
t is a transition; this transition holds the logical predicate of the transition.
altt is an alerter to monitor the condition at transition t; altt will be triggered whenever the transition is fired (active and enabled).
($pi_1 \ldots pi_m$) are the input places for the transition.
($po_1 \ldots po_l$) are the output places for the transition.

In-type has the following form:

(t,{[pi_1, ni_1, $colori_1$],....[pi_j, ni_j, $colori_j$]})

where:
t is a transition.
[pi_k, ni_k, $colori_k$] indicates, when transition t is fired, ni_k tokens with color $colori_k$ will be removed from its input place pi_k.
j is the number of input places of transition t.

Out-type has the following form:

(t, {[po_1,no_1, $coloro_1$],...,[po_j,no_j,$coloro_j$]})

where:
t is a transition.
[p_{ok}, n_{ok}, $color_{ok}$] indicates when transition t is fired, n_{ok} tokens with color $color_{ok}$ will be added to its output place p_{ok}.

Finally, associated with each D-type knowledge table there is a marking vector that determines the distribution of the tokens in the net. This vector has the form (u_1, u_2,... u_n) where n is the number of places, each u is a pair (w, n_w), w indicates a "white" token, and n_w the number of white tokens in the place. As the only function of a black token is to reset the number of white tokens at a place to zero, information about black tokens is not included in the marking vector [7].

Active and enabled transition vectors are needed also to control execution of the transitions. The active transition vector is $(V_1,V_2,....V_m)$, where V_i is a predicate that indicates if transition t_i is active, and m is the number of transitions in the net. An enabled transition vector is used to indicate which transitions are enabled. It has the form $(e_1,e_2,e_3...e_m)$, where e_i is a boolean value indicating whether transition t_i is enabled (t_i is enabled if there is at least one white token in each of its input places). This vector has a different interpretation for static knowledge representation.

In the simple Petri net presented in figure 4 the tokens (dots in the circles) in places p1, p5 and p7 can be considered white; so that the marking vector is:

u = {(w,1), (w,0), (w,0), (w,0), (w,1), (w,0), (w,1)};

the active transition vector, for the specific state of the net, could be:

v = (0, 1, 1, 0, 1, 0),

this vector cannot be inferenced from figure 4 because v_i is a predicate that indicates whether transition t_i is active; the enabled transition vector for the same net is:

e = (0, 1, 0, 0, 0, 0).

This means that, in the current state of the system, only transition t_2 can be fired, because v_2=1 and e_2=1.

More details about the knowledge slices (i.e. Place-type, Trans-type, In-type, Out-type), alerters, active transition vector, etc. are presented in Chapter IV (Implementation Issues).

In summary, for representing a D-net as a knowledge table it is necessary to consider at least the following data structures:

Knowledge table name

Marking vector $U = (u_1, u_2 \ldots\ldots u_n)$

Active transition vector $V = (v_1,\ v_2 \ldots\ldots v_m)$

Enabled transition vector $e = (e_1,\ e_2 \ldots e_m)$

Place-type slice:	One place related with all its input and output transitions.
Trans-type slice:	One transition related with all its input and output places.
In-type slice:	One transition related with all its input places and the number of white or black tokens associated with it.
Out-type slice:	One transition related with all its output places and the number of white or black tokens associated with it.

Features and Advantages of D-net Model

The D-net model which is derived from the G-net model, combines the features of Petri nets and knowledge tables; those features and advantages are presented in Chapter I, where the model is introduced.

Reasoning for Dynamic Nets

The occurrences of events are the bases for the reasoning strategies and algorithms for dynamic knowledge represented by a D-net. As in any reasoning process, the objective is to establish subsequent activities or processes based on the current state of the system (i.e. the current state of the D-net). The initial marking of the D-net is determined by the initial state of the system being represented.

The general guideline of the reasoning process is as follows: when a transition t is fired, the number of tokens in the input places p_{ij} will be decreased by I(t) (the input function for that transition), and the number of tokens in the output places p_{oj} will be increased by O(t) (the output function for that transition). When a black token is added to a place p_i, the number of white tokens is set to zero and the white token is removed. In this way the activities in those output places are disabled. When a token is added to or removed from a place p, the alerter altp2 is triggered to take corresponding action. When a token is removed, the alerter altp2 invokes a procedure to check if it has to disable transitions t_{o1}, t_{o2}, ... t_{oq}; this can be done by updating the enabled transition vector. When a token is added to a place p, the alerter altp2 invokes a procedure to check if any of the transitions t_{i1}, t_{i2}, ... t_{ip} becomes enabled, according to the information of the marking vector.

Complex D-nets and Control Table

A knowledge object may be an atomic object or a complex structure. A complex D-net is a D-net in which one or more places or transitions are D-nets themselves. To construct a complex D-net it is necessary to provide a mechanism to connect elementary D-nets together.

Deng and Chang [7] introduce a special place called a "switch place" to connect elementary D-nets. A switch place is defined by the quintuple (SID, K, u, ETV, ATV), where:

SID is a unique name for the net that the switch represents,
k is the same k of the D-net, which can be "static" or "dynamic",
u is the marking vector of the net,
ETV is the enabled transition vector, and
ATV is the active transition vector of the net.

The connections among D-nets are through the switch places. To connect two nets D1 and D2 we can attach one or more transitions in D1 to the switch place of D2, and so on. Once tokens from one D-net enter the switch place of another D-net, they enter another D-net level. Figure 7 shows an example of a Complex D-net. In a complex D-net, when the reasoning process jumps from one level to another, the control should come back to the point where the jumping took place when the reasoning at that current level is done; the SID field of the switch is used to know where the jump should be done.

Formally a complex D-net is a D-net composed of a set of elementary or complex D-nets connected to each other through switch places. Theoretically, any place can be connected to a switch through a transition, and any connection can be broken by setting the condition on the transition to false; therefore the composition of a complex D-net can be modified dynamically.

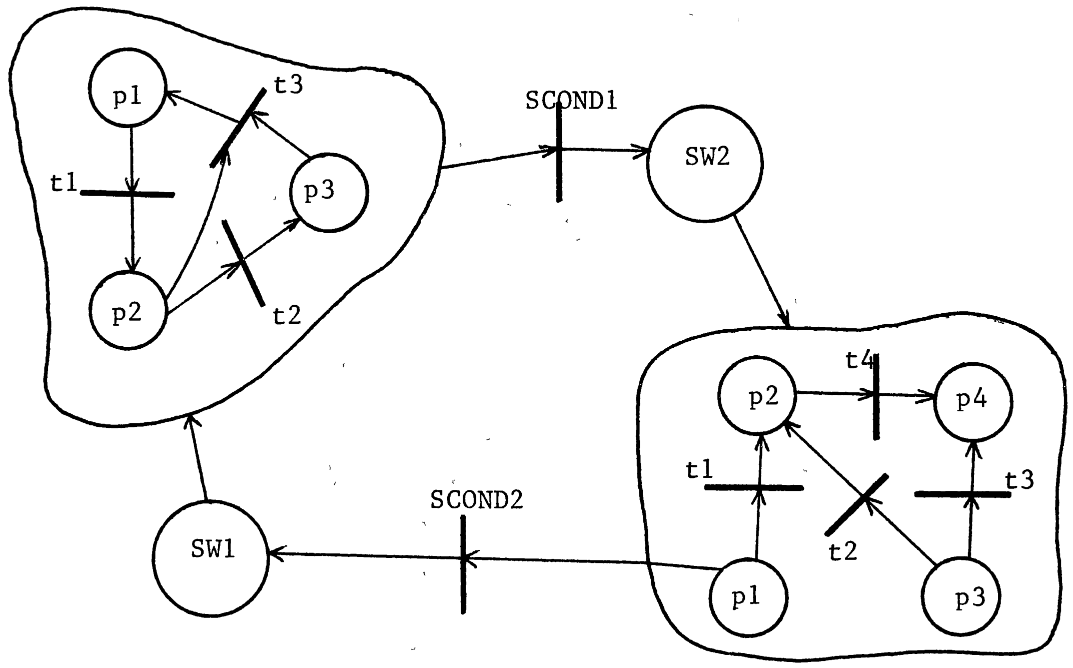

Figure 7. A Complex D-net

For an elementary D-net the knowledge table representation is used to perform the reasoning. The "control table" is used to perform reasoning on a complex D-net. The most important function of the control table is to specify how the reasoning process should start.

A control table is composed of a set of control records with one record per switch place in the D-net. Each control record is defined by a set of quintuples:

<SID, ISET, SCONN, SCOND, SPEC>

where:

SID	=	Unique identification for a switch place.
ISET	=	Initial setting for the switch place. Includes: marking vector, enabled and active vectors.
SCONN	=	Connections of the switch place with another elementary D-net.
SCOND	=	Condition to enter to switch place.
SPEC	=	Specifies the termination states(s) of the D-net.

The contents of the control table can be updated in a manner similar to updating a knowledge table. The tasks of the control table reasoning is to start a reasoning process, coordinate and select the appropriate subsequent places or subnets at different stages of a reasoning process and dynamically change the structure of the complex D-net when necessary.

Reasoning on a complex D-net must start from one of its elementary nets, called the current part. This current part is actually the net in which the reasoning process is being done. During the process of reasoning one of the following three events can occur to transfer control back to the control table:

a) if during reasoning the contents of the control table are updated, the control table algorithm should be invoked to adjust the initial default settings, conditions and connections.

b) if a token reaches a switch place, control will pass back to perform the reasoning in the current part.
c) when the reasoning in a particular part is done, the control table algorithm must pass the results and control back to the place from which the current part was entered.

The complex D-net is designed to model complicated asynchronous activities because a big set of tasks can be subdivided in smaller subsets or modules and then each subset modeled independently; after doing so the modules or abstract entities can be integrated to construct the complex D-net.

CHAPTER IV

IMPLEMENTATION ISSUES

Details about the implementation of the algorithms for reasoning with Petri nets and Knowledge Table are presented in this chapter. Some specific rules that should be considered during the design of a dynamic system also are presented.

The prototype for reasoning was developed in C language using the Turbo C compiler version 2.0 in an IBM-compatible microcomputer [22]. During the testing and debugging stages some subroutines were written to show partial results of reasoning. The final program occupies about 80 kbytes and contains 35 procedures.

Data Structures

The data structures chosen to represent the places, transitions, tokens and connections of a net are some of the most important factors for successful implementation. The information stored in the data structures should give complete information about the places, transitions and tokens in order to obtain accurate and fast results [14].

Places

Each place has fields for:

- Place name. A unique name or number for the place.

- Place Type. A place can be a value, an activity or a switch. If the place is a value, it can store several values; an activity is an external program that performs certain calculations and returns the results to the net using a file. A switch is used to connect different subnets, so that a switch cannot contain activities.

- Activity name. Name of the external program that is invoked when the reasoning process reaches the place.

- Initial Tokens. Number of tokens that the place contains at the beginning of the process. The user should specify the initial tokens as the initial setting of the system.

- Number of Tokens. Number of current 'white' tokens in the place. This number changes according to the reasoning process.

- Transitions Input. This is a vector that contains the names of the transitions that have current place as an input place.

- Transitions Output. This is a vector that contains the names of the transitions that have current place as an output place.

- Alerter. This field is to indicate that the activity of the place can be executed by the system, so the place is put into the execution queue.

- Executed Flag. This flag indicates that the activity of the place was executed.

- Event. The event can be 'adding tokens' or 'removing tokens' from the place. The type of event is used to control the reasoning process.

- Results. Vector that contains the results of the place. This is an array of strings where each string can contain an integer, a float or a string of characters. For the reasoning algorithm all the variables are strings because the program does not have to do any calculations with the data of the system being analyzed.

Transitions

Each transition has fields for:

- Transition Name. A unique name or number for the transition in the net.

- Condition. Initial condition of the transition. If condition is equal to '0' the transition is an activity that should be executed by an external program. If it is equal to '1' the transition is 'static', so no external program is called. When the transition is an activity the reasoning process calls the corresponding program taking as arguments the results of the input places of the transition. When the transition is static the results af every input place are passed to the result vector of the transition, the order is the same as specified in the list of input places of that specific transition.

- Activity Name. Name of the activity or external program that is invoked when the transition is enabled.

- Alerter. The alerter indicates that the activity of the transition can be executed by the system, so the transition is put in the execution queue.

- Input Places. Array that contains the names of the input places of the transition.

- Input Tokens. Array that contains the number of tokens that must be removed from the input places when the transition is fired.

- Color of Input Tokens. Array that contains the color of every input token. This implementation uses only 'white' and 'black' tokens as is specified in the description of the D-net, chapter III.

- Output Places. Array that contains the names of the output places of the transition.

- Output Tokens. Integer array that contains the number of tokens that must be added to the output places when the transition is fired.

- Color of Output Tokens. Integer array that contains the color of every output token. This implementation uses only 'white' and 'black' tokens.

- Results. Vector that contains the results of the transition. As in the results of the places, this is an array of strings that can store integers, reals or characters strings.

Current Alerter

The current alerter is a place or a transition that is taken from the execution queue. This data structure contains the following fields:

- Alerter Name. Name of a place or transition.
- Type. Indicates whether the alerter is a place or a transition

Active Transition Vector

This is a single vector of integers that indicates whether the transition has been executed.

Enabled Transition Vector

The enabled transition vector is used to indicate that a transition can be fired according to the Petri net rules.

Execution Queue

This is a circular queue that contains the names of the places and transitions to be executed. The sequence stored in this queue is not necessarily the execution sequence of the system being analyzed, it is only the sequence given by the algorithms in order to check if those nodes can be executed, according to the rules of Petri nets.

Files

Four different files are used during the reasoning process: the input file, two files to store temporarily the results of activities performed by places and transitions, and one output file that contains the trace of the execution of the reasoning process.

Input File

The design of the input file is presented in figure 8, and in chapter V there are examples of the input file. The conventions used in figure 8 are as follows:

- The keywords can be in upper or lower case letters and they are preceded by one or two starts '*'. The keywords are '**PLACES', '*TI', '*TO', '**TRANSITIONS', '*PI' and '*PO'.

- The names used by the user are in lower-case letters; for instance 'nplaces' and 'type-of-place' indicate the number of places of the system and the type of place respectively.

- The optional parameters are indicated between square parenthesis "[]", for example '[name of activity]' is an optional parameter that depends on the type of place.

- The name of places and transitions can contain a maximum of 8 characters.

Now, a brief description of every parameter specified in figure 8 is given, starting with the keywords.

- **PLACES: Indicates the beginning of the place descriptions.

- *TI: Transition Input; indicates the transitions that have the current place as one of their input places.

- *TO: Transition Output; indicates the transitions that have the current place as one of their output places.

- **TRANSITIONS: Indicates the beginning of transitions descriptions.

- *PI: Places Input; places that are input to a transition.

- *PO: Places Output; places that are output from a transition.

*PO and *PI constitute the set of transition functions of the system.

- name-of-the-system: Name of the system that is being represented as a Petri net; maximum length is 80 characters.

- n-places: Number of places in the Petri net.

- place-1 ... place-n: Name of places of the system.

- place-type: Type of place that can be 'v'=value, 'a'=activity, 's'=switch.

- init-ntokens: Initial number of tokens of the place, usually is zero(0).

- [activity-name]: This parameter is optional; it is specified only when place-type is 'a' or init-cond is '0'; it should be the name of an executable program that is called when the place is activated.

- trans-i1 ... trans-ip: Name of transitions that have the current place as one of the input places.

- trans-o1 ... trans-oq: Name of transitions that have the current place as one of the output places.

- n-trans: Number of transitions of the Petri net.

- trans-1 ... trans-t: Name of transitions of the system.

- init-cond: Initial state of the transition. The possible values are '0' and '1'. init-cond='0' means that the transition is not active and should be executed by an external program specified in [activity-name]. init-cond='1' means that the transition is active and is only a connection among places, so that the result of the transition is the set of results of its input places.

- (p-i1 ntok-i1 color-i1) ...(p-im ntok-im color-im): This set of parameters specifies the input function of a transition. The interpretation of this function is: when a transition is fired ntok-ik tokens of color-ik are removed from the input place p-ik.

- (p-o1 ntok-o1 color-o1) ...(p-om ntok-om color-om): This set of parameters specify the output function of a transition. When a transition is fired ntok-ok tokens of color color-ok are added to the output place p-ok.

```
name-of-the-system
**PLACES  n-places
place-1 place-type init-n-tokens [activity-name]
*TI   trans-i1 trans-i2 ....
*TI   ........ trans-ip
*TO   trans-o1 trans-o2 ...
*TO   ........ trans-oq

place-2 place-type init-ntokens [activity-name]
*TI   . . .
*TO   . . .

place-n . . .
. . .

**TRANSITIONS  n-trans
trans-1 init-cond [activity-name]
*PI   (p-i1 ntok-i1 color-i1) (p-i2 ntok-i2 color-i2) ..
*PI   ...............(p-im ntok-im color-im)
*PO   (p-o1 ntok-o1 color-o1) (p-o2 ntok-o2 color-o2) ..
*PO   ...............(p-ol ntok-ol color-ol)

trans-2 init-cond [activity-name]
*PI   ...
*PO   ...
. . .
. . .

trans-t . . .
. . .
```

Figure 8. Design of the Input File

Output File of Transitions and Places

The results of transition and place execution are stored in a file that is used as an interface between transitions and places. When an activity is executed, its results are put in the file. Later the reasoning process takes those results to stores them in the vector of results for the corresponding transition or place. In this way the activity of a node of the system is independent of the reasoning process, and its results are handled by the reasoning process without interpretation.

Figure 9 presents the design of the output file of a transition, and figure 10 shows the structure of the corresponding file for a place. In the figures, 'npar' is the number of parameters or results produced by an activity; 'val1', 'val2' ... 'valn' are results that can contain integers, reals or strings. Finally, 'cond' is a parameter for transitions that indicates if the activity was successful.

```
npar
cond  val1 val2 val3 ...... ..... valn
```

Figure 9. Structure of Output File of Transitions

```
npar
val1 val2 val3 ................ valn
```

Figure 10. Structure of Output File of Places

Trace File

The details of the reasoning process are stored in the trace file. This file was designed to help the user to follow and analyze the results obtained during the execution of the system.

The trace file contains the specifications given in the input file (whose design is given in figure 8), the results of every place and transition execution, snapshots of the execution queue and the actual execution sequence of activities. The order and meanings of the partial results or "messages" of places and transitions are the responsibility of the designer of the system.

Results of Event-driven Reasoning

The goal of the reasoning process is to determine subsequent events or activities based on the current events of the system [7]; therefore, the result of reasoning is the logical execution of the activities represented in places and transitions. In chapter V, the execution sequences of the systems, used as examples, are presented. These sequences show the partial results of every place and transition; in this way the user can follow and analyze the different execution sequences and results of the system being studied.

Special Considerations and Limitations

The program for event-driven reasoning was developed as a prototype to study the behavior of the algorithms in different situations; however, this software can be used for designing information systems and for studying their behavior under different conditions or situations.

The software considers the rules of Petri net theory and also includes some specific rules that should be taken in account during the design of a net. Those special considerations are:

- The name of places and transitions should be unique in the whole system. In other words, the name of a node in a subnet should not be repeated in another subnet.

- A switch place is used to connect subnets, pass the results from one set of subnets to another and to reset the conditions of subnets. The switch place cannot be used as an activity in the system. The transitions associated with a switch place can contain activities.

- There is a file for communication between places and transitions. Although this is adequate for the prototype, for some applications or expert systems it could be necessary to use more files when the number of results is high and should not be stored in main memory (vector of results in the program).

- The input arguments for the activity of a transition is the ordered set of results of every input place of the transition. In the same way, the input arguments for the activity of a place is the ordered set of results of every input transition to the place. Special care should be taken when a result is missing during the execution of an activity, because the reasoning calls the external activities using all the results that are available in the precedent nodes.

- To start the reasoning the first alerter should be one which is active. The first alerter can be a place, a transition or a switch. Sometimes is convenient to use a special place or transition dedicated as the starting alerter.

- The system to be analyzed can be designed as a cycle (a closed net) if the user wants to see its behavior with different values or conditions. Before starting a new cycle the reasoning process ask the user if he wants to run another cycle of the system.

- The names of places, transitions and keywords, in the input file, can be written in upper or lower case.

- To connect different subnets the user should indicate the specific connections between the switch place and the transitions of the subnets.

CHAPTER V

ANALYSIS OF RESULTS

Example 1: A Prototype of an Expert System for Lime Recommendations for Acid Soils

In order to test how the reasoning process can be applied to a sequential system, the procedure to do lime recommendations for acid soils was chosen.

The soil is constantly changing its chemical characteristics; it is like a big chemistry laboratory. As a consequence of nitrogen fertilization, removal of Calcium, Magnesium, Potassium, Sodium and other basic cations, and the decomposition of organic residues, the soil can be acid (pH: 3.0-7.0), and this acidity severely affects crop production, especially for certain crops that are less resistant to acidity [1], [29]. To neutralize acidity in soils it is necessary to apply lime (Calcium Carbonate) to it; the lime induces chemical transformations that modify the soil pH, from acid to basic, and then the crops can absorb nutrients more effectively.

This expert system is a prototype to estimate lime recommendations. There are 2 general situations: in the first situation there is sufficient historical information on the region to obtain statistical models of pH based on other variables that are also chemical determinations; in the second situation there is no information. In the second case the lime recommendation can be made theoretically using other statistical models based only on the type or texture of soil. With the historical data base, which contains soil analysis data, the models for region and texture can be obtained to feed the knowledge base; the data base is updated constantly because new soil analyses are added to it, and periodically new, more accurate models can be obtained.

The prototype was developed according to the structure shown in figure 11. The statistical models for this prototype were not obtained with real data from the regions; they were obtained according to the general model of acidity (pH) in the regions and textures utilized [30].

Description of the System

A general description of the system is presented in figure 11. A soil can be basic, neutral, or acidic; a soil is basic when its pH is greater than 7.0, and it is acidic when its pH is below 7.0. pH is a chemical determination defined as the negative log of concentration of ions of hydrogen(H):

$$pH = -\log(H^+)$$

The acidity of soils affects crop production considerably, and it is necessary to apply lime in order to obtain an appropriate soil near neutral to cultivate different crops [29].

The acidity of soil affects crops in different ways depending on the type of soil and type of crop; therefore, obtaining the correct amount of lime to be applied is not an easy task; agronomists must take into account many variables to calculate the necessity of lime. The results obtained from chemical calculations and soil tests are not always accurate because the soil is constantly changing its characteristics and is not uniform; there are so many types of soil, textures, chemical elements and different environments that is difficult to get the ideal recommendation of lime. Another problem is that the samples of soil analyzed in laboratories are only small amounts.

Lime requirements and recommendations, as determined by soil tests, are made in terms of effective calcium carbonate equivalent(ECCE) which is a quality standard based on the percent purity and fineness of material; as a result, lime recommendations and applications should be based on this parameter. Other important factors that should be considered in lime recommendations are desired pH for crop species, depth of soil to be tested, rate of reaction of lime according to type of soil with different characteristics, and environmental conditions. The efficiency of other fertilizer nutrients is improved as the limed soil approaches neutral pH [30].

The main idea behind this expert system prototype is to combine the statistical analysis provided by a historical data base on soil analysis and chemical calculations to obtain a good lime recommendation; in addition, the experience of a human expert, such as an agronomist, should be considered for the final recommendation [30].

The knowledge for developing this prototype was provided by Dr. Westerman, a professor of Agronomy at O.S.U. [30], and some books related with soils that are listed in the bibliography [1], [29].

The expert system, when fully developed, will be useful for agronomists, farmers, producers and researchers of soil sciences because it can help to improve yield in several crops. This prototype was developed for doing lime recommendations in one crop, a few regions and a few soil textures; however, it can be extended to different crops regions and textures; obviously, the system will be more complicated, and more data will be needed to obtain the statistical models, by region and texture or any other variable that might be important in the models.

The data used to obtain the statistical models for this prototype was not taken from real soil analyses, therefore the lime recommendations are probably not accurate and the user should not use the results of this prototype as a precise lime recommendation.

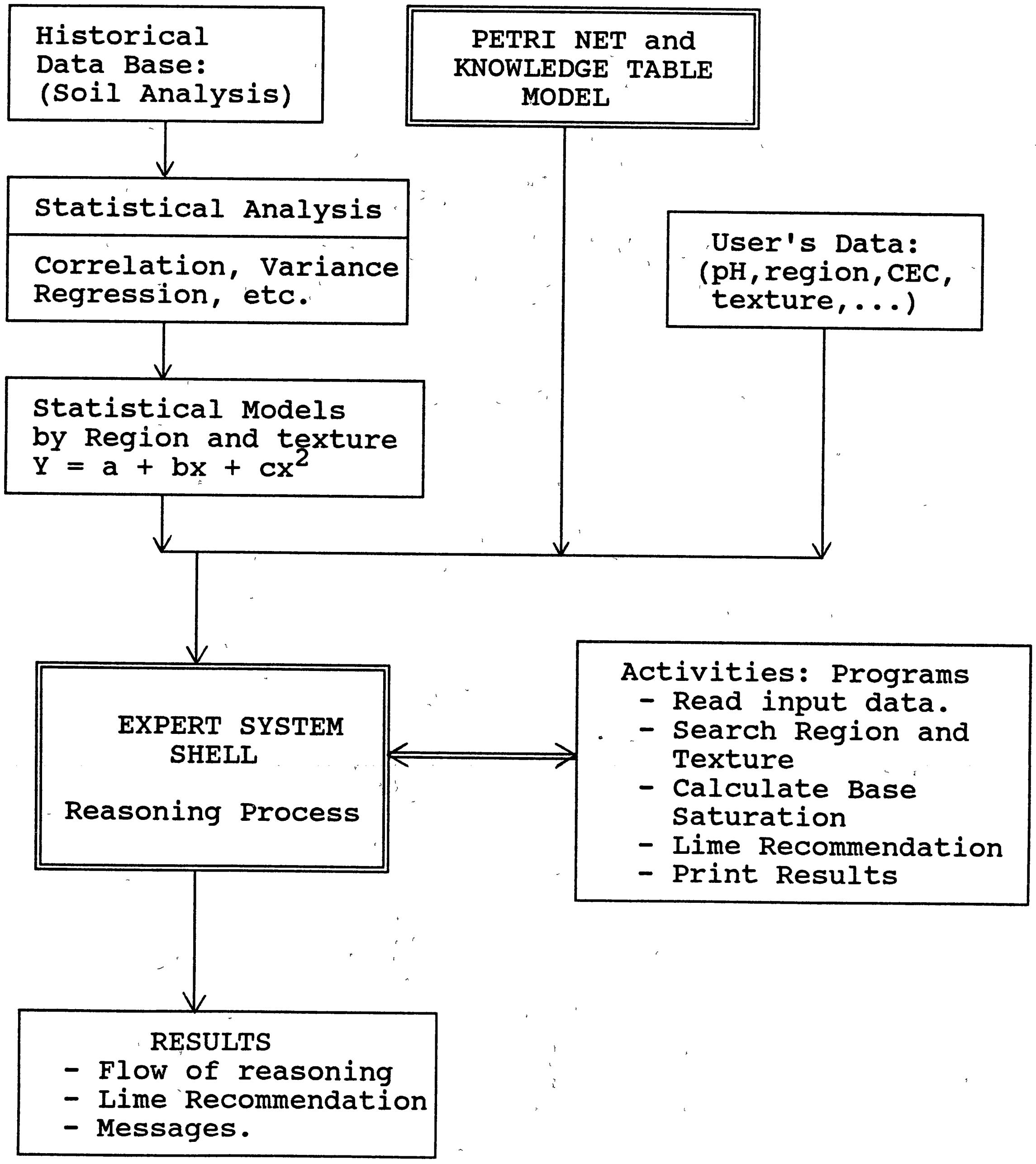

Figure 11. General Description of the System Prototype for Lime Recommendations

The prototype has two basic inputs: a historical data base and the data issued by the producer or researcher about soil and crop.

The historical data base contains soil analyses classified by region and texture; based on this data the statistical models used by the expert system are obtained. In the real application this data base is very important for the system because it contains detailed information about the soil.

The producer or user should enter data related to soil and crop in order to choose the appropriate model for the specific soil. The user should enter the following information requested by the Expert System:

REGION : Name of the region in which the crop is located. The system presents the available regions and then the user selects one.
TEXTURE: Type of soil texture. The system presents the available textures and the user selects the appropriate one.
BASE SATURATION DESIRED: Percentage of Base saturation that the user wants for the soil. Base saturation is a chemical determination.
pH : Acidity or basicity of soil. Typical values can be in the range (3.0 - 8.0).
CEC : Cation Exchange Capacity [meq/100g]. This is a chemical determination whose units of measure are miliequivalents per 100g of soil. Usually the range is (0.0 - 40.0).
ECCE: Effective Calcium Carbonate Equivalent. This is an index of purity and fineness of lime and its value is usually in the range (0.5 - 1.0).

If the region and texture given by the user are in the data base, the expert system takes the corresponding model and the recommendation is done with that model. If the region is not in the data base, the expert system does a theoretical recommendation based on texture and other input data.

According to region and texture there are three possibilities to obtain the statistical model:

1. The region and texture are in the models; so the expert system can obtain a real model.
2. The region given by the user does not contain the desired texture; so, obtain a theoretical model.
3. When the user selects region = "other" and any texture, the system calculates lime with a model based only on texture.

Petri Net and Knowledge Table

Model of the System

The Petri net developed for this prototype is presented in figure 12; the net is a closed net, so that the system loops requesting more input data for different cases until the user stops it.

Activities or External Programs. The activities for this prototype are external programs or functions that perform the corresponding activities established in the Petri net of the system. Every activity performs specific tasks; below there is a description of each activity.

LRREAD is a function that allows the user to enter the input data requested for the prototype.

LRIF1 is a program whose function is to define if there is a model for region and texture.

LRIF2 has the same function as LRIF1, but LRIF1 returns

'1' as a token when the activity is successful while LRIF2 returns '1' when the activity is unsuccessful.

LRBSREAL is a program to calculate 'base saturation' according to the statistical model found (determined by region and texture).

LRBSTH calculates 'base saturation' based only on texture. This activity is called only when transition LRIF2 is true.

LRBSNEC is a function to calculate the 'base saturation necessity' of the soil.

LRIF3 establishes if 'base saturation necessity' is greater than zero; if it is, then that transition is true.

LRIF4 has the same function as LRIF3, but LRIF4 is true if 'base saturation necessity' is less that zero.

LRLIME is the program that calculates the final lime recommendation according to the information sent by LRIF3.

LPRINT is in charge of printing the results of the prototype.

T1, T2, T3 and T4 are transitions whose only function is to connect places and pass information(tokens) from the input places to the output places. These places are always active because they do not need external procedures to execute their function.

The final advice given by the expert system is the amount of lime that should be applied to the soil; this value is given in pounds per acre [lb/acre].

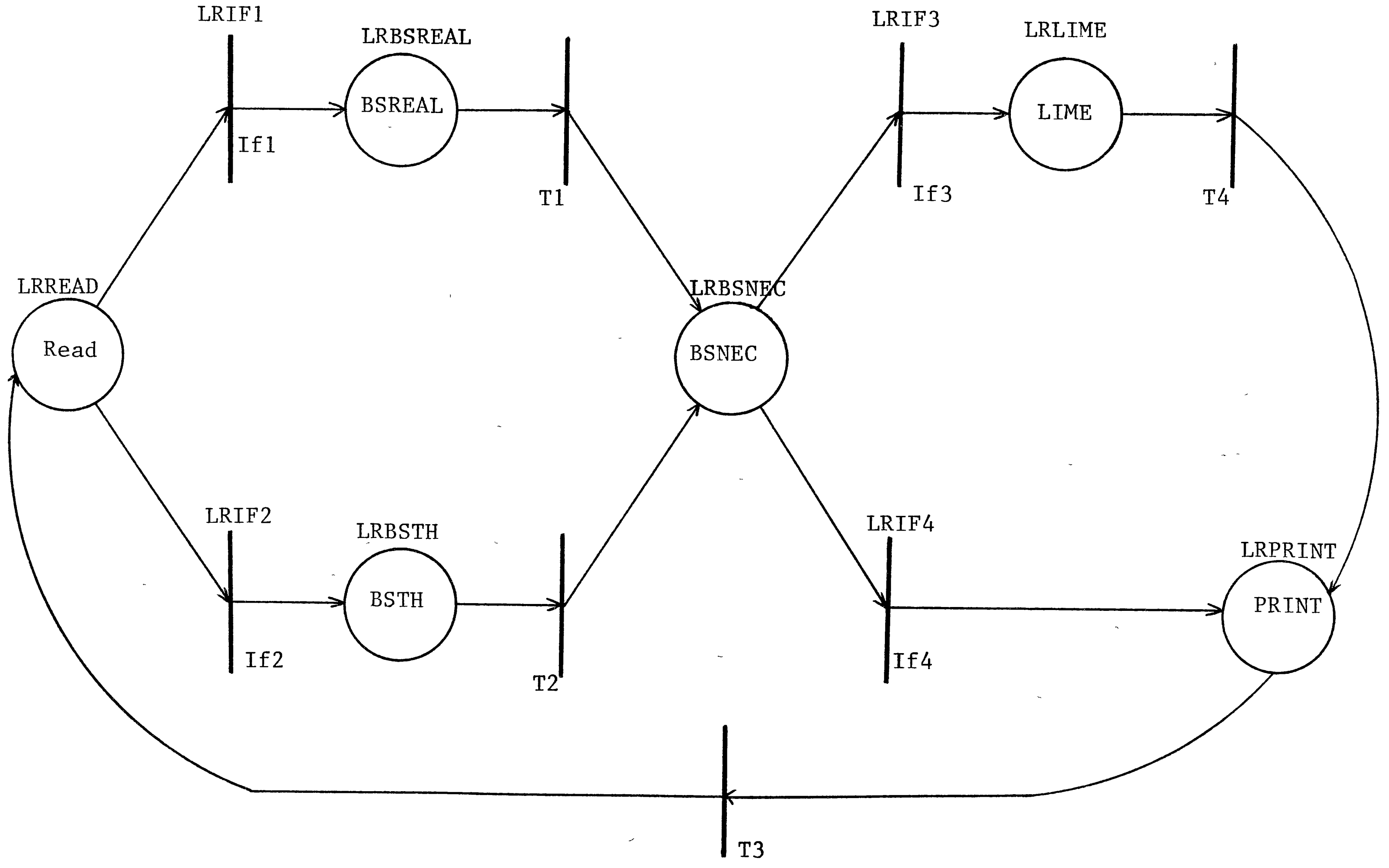

Figure 12. Petri Net of Lime Recommendation Prototype

Input File for the Reasoning Process. Figure 13 presents the contents of the input file used for this prototype. This file contains the specification of the Petri net in combination with the knowledge table representation, and it can be created following the specifications given in chapter IV.

Results of Lime Recommendation System Prototype

The results of the reasoning process combined with the results of lime recommendations are presented in Appendix A; the specific screens created by the activities are not included. The log of reasoning shows the partial results obtained in every place or transition; the partial results and the order of those results in every node are determined by the designer of the Petri net representation of the system.

It should be emphasized that the input used to generate the statistical models were not from real data. Therefore, the procedures outlined and results obtained are for illustrative purposes only.

The design of this sequential system is flexible, and the user can modify the activities in a node without changing the original design.

Sometimes it is necessary to include nodes (especially transitions) that are not clearly defined in the system to

```
Lime Recommendation for Acid Soils.
**PLACES 6
Read a 0 LRREAD
*TI  if1 if2
*TO  t3
bsreal a 0 lrbsreal
*TI  t1
*TO  if1
bsth a 0 lrbsth
*TI  t2
*to  if2
bsnec a 0 lrbsnec
*TI  if3 if4
*to  t1 t2
lime a 0 lrlime
*TI  t4
*TO  if3
print a 0 lrprint
*TI  t3
*to  t4 if4
**TRANSITIONS 8
if1 0 lrif1
*PI  (read 1 1)
*PO  (bsreal 1 1)
if2 0 lrif2
*PI  (read 1 1)
*PO  (bsth 1 1)
t1 1
*PI  (bsreal 1 1)
*PO  (bsnec 1 1)
t2 1
*PI  (bsth 1 1)
*PO  (bsnec 1 1)
if3 0 lrif3
*PI  (bsnec 1 1)
*PO  (lime 1 1)
if4 0 lrif4
*PI  (bsnec 1 1)
*PO  (print 1 1)
t4 1
*PI  (lime 1 1)
*PO  (print 1 1)
t3 1
*PI  (print 1 1)
*PO  (read 1 1)
```

Figure 13. Input File for Lime Recommendation Prototype

establish connections; this happens with transitions t1, t2, t3 and t4 (see figure 12), for instance, t3 is used to restart the system after every lime recommendation.

The user or designer can follow the execution sequence of the system and check all the details in order to debug the design; this prototype shows the sequence of execution of places and transitions.

In this system the data passed from node to node is not much; however, when the amount of data is high it would be better to establish another strategy for the tokens, for example utilizing several files or one dynamic file to store the partial results without passing data from node to node.

Using the Petri net representation one can observe the different paths of the execution sequence according to the results of every node; in this system there are four possible paths that can be distinguished in figure 12. This is an important point for the designer because he/she can test the system by forcing it to take different paths using special conditions or data.

An important feature of this technique is that the designer can isolate the activity of every node like a function in a programming language with input parameters (messages) and output or results; usually these results are used as input data by the consequent nodes.

To include or delete a place in the net, normally it is also necessary to modify a transition because the places and transitions are mutually dependent. This aspect is observed more often in sequential systems like the present prototype.

Example 2: Arithmetic Computations as Concurrent and Asynchronous Activities

A set of arithmetic computations that can be executed concurrently (in parallel) was selected to test the prototype with a system that exhibits concurrency and whose activities can be performed in an asynchronous form.

The main objective of this example is to show how the activities of the places depend only on their precedent transitions, and the activities of the transitions depend only on their precedent places. A set of activities can be executed concurrently and asynchronously with another set of activities of the system when they are independent (executed in different branches of the Petri net).

Petri Net and Knowledge Table of the System

Figure 14 presents the Petri net used for representing the concurrent arithmetic calculations. The objective of this net is to obtain the result of the expression

$$x = (a+b) / (a-b);$$

once the system reads the values of 'a' and 'b' the expressions (a+b) and (a-b) can be evaluated concurrently because they are independent; for instance, the set of nodes {p1, copya, p3, p4} can be executed in parallel with the set

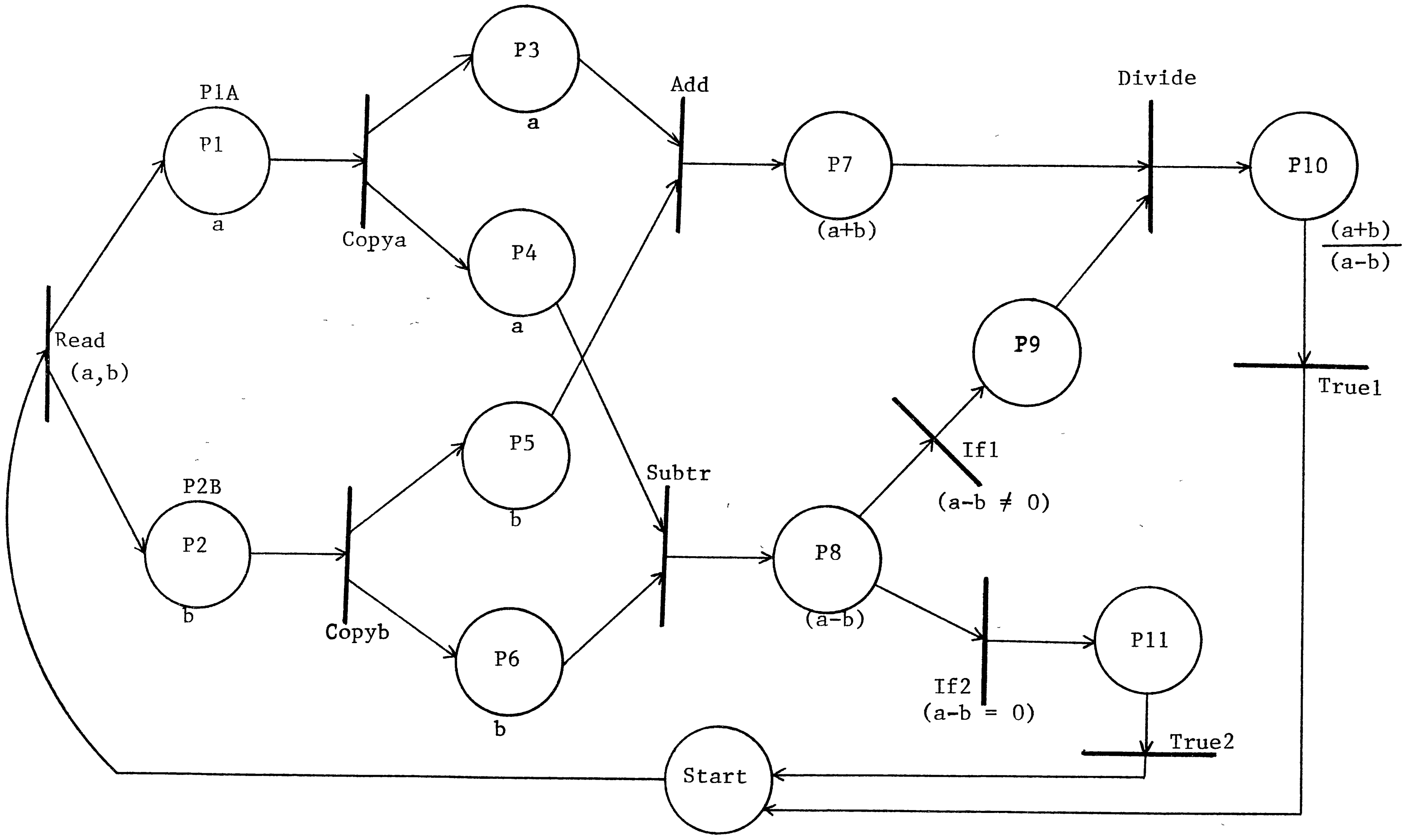

Figure 14. Petri Net of Concurrent Computations

of nodes {p2, copyb, p5, p6}, the node 'Add' can be executed concurrently with 'p6', 'Subtr.', 'p8' and so on.

The place 'start' is used to initiate the system. It also could be used as a switch place; during the tests this place was used as initial alerter and switch place obtaining the same results. The Petri net of this system is closed (a loop) in order to facilitate continuous calculations with different values entered by the user.

In this example most of the activities of places and transitions are small external programs that are called by the reasoning process according to the logical flow of the system; the activities are:

READ: This is a function (transition) that asks the user to enter two values 'a' and 'b'; these values can be integer or real;

P1A: The function of this place is to store the value of 'a' that was read in the previous transition.

P2B: This function is in charge of storing the value of 'b' that was read in the transition 'READ'.

ADD: This transition adds the values of 'a' and 'b' which are taken from the places 'p3' and 'p5'.

SUBTR: This transition finds the difference between 'a' and 'b', which are taken from places 'p4' and 'p6'.

IF1: This function checks if (a-b) is different from zero; if (a-b) is different from zero a token is sent to place 'p9' and the process continues.

IF2: This program checks if (a-b) is equal to zero; if (a-

b) is equal to zero a token is sent to place 'p11' and the process continues in that branch. Therefore the places 'p9' and 'p11' are mutually exclusive.

DIVIDE: This program divides the result of place 'p7' by the result of place 'p9'.

The rest of the nodes {copya, copyb, p3, p4, p5, p6, p7, p8, p9, p11, p10, true1, tru2, start} do not need an external program to execute their activity, which is basically to transmit tokens.

Figure 15 presents the contents of the input file of the system that was used for this example. This file is the representation of the Petri net as knowledge table in order to execute the reasoning process.

Results of the Concurrent Computations Prototype

The results of the reasoning process are presented in Appendix B together with the results and messages of the programs or activities of every node. In this trace all the results produced by every place and transition are presented in order to allow the user to follow the different execution sequences of the system as well as to observe the activities that can be performed concurrently.

The basic execution sequence of this prototype is: start, read, p1, copya, p2, copyb, p3, p4, p5, add, p6, subtr, p7, p8, if1, if2, p9, divide, p10, true1,...; in this basic sequence the nodes p11 and true2 are not included

```
Mathematic Computations
**PLACES 12
p1 a 0 p1a
*TI  copya
*TO  Read
p2 a 0 p2b
*TI   Copyb
*TO  Read
p3 v 0
*TI  Add
*TO    Copya
p4 v 0
*TI  Subtr
*TO  Copya
p5 v 0
*TI  add
*TO  Copyb
p6 v 0
*TI  Subtr
*TO  Copyb
p7 v 0
*TI  Divide
*TO  Add
p8 v 0
*TI   If1 If2
*TO  subtr
p9 v 0
*TI  Divide
*TO  If1
p10 v 0
*TI  true1
*TO  Divide
p11  v 0
*TI  true2
*TO  If2
start s 1
*TI  read
*TO  true1 true2
```

```
**TRANSITIONS 10
Read 0 read
*PO  (p1 1 1) (p2 1 1)
Copya 1
 *PI   (p1 1 1)
 *PO   (p3 1 1) (p4 1 1)
Copyb 1
 *PI   (p2 1 1)
  *PO   (p5 1 1) (p6 1 1)
Add 0 add
*PI  (p3 1 1) (p5 1 1)
*PO  (p7 1 1)
 Subtr 0 subtr
*PI    (p4 1 1) (p6 1 1)
*PO    (p8 1 1)
Divide 0 divide
*PI    (p7 1 1) (p9 1 1)
*PO    (p10 1 1)
 If1 0 if1
*PI    (p8 1 1)
*PO    (p9 1 1)
If2 0 if2
*PI    (p8 1 1)
*PO    (p11 1 1)
True1 1
*PI    (p10 1 1)
*PO    (start 1 1)
True2 1
 *PI    (p11 1 1)
*PO    (start 1 1)
```

Figure 15. Input File of Concurrent Computations

because if2 does not send a token to p11 when (a-b) is positive.

The activity of a place is executed as soon as the precedent transition is fired and sends a token to it; for instance, p1 is executed immediately after the execution of transition read, p3 and p4 are executed immediately after the execution of transition copya. The activity of a transition is executed as soon as all its precedent places are executed; for example, transition Add is fired immediately after places p3 and p5 are done.

In this example the reasoning attempts to execute places p3, p4, p5 and p6 in order; however, when place p5 is executed all the precedent conditions to Add are satisfied, so transition Add is fired before executing the activity of place p6.

This prototype shows that activities can be executed concurrently and in an asynchronous way; in other words the timing of the activities is not considered. For example, to fire the transition divide it is necessary that all the activities of the two precedent paths had been executed. In this case the reasoning process does not consider the duration of the activities.

The order of the execution sequence can easily be modified. This can be done changing the order of the parameters in the input file so that the original design does not have to be modified. The reasoning process is very sensitive. Any change in the knowledge table or in the Petri net model is detected and the sequence of the system could be altered. This aspect is important, especially for the designer of a system, because he/she can observe the behavior of the system being simulated under different situations by changing certain parameters or conditions.

Analysis of Algorithm for Complex Nets

Using switch place to connect elementary nets is useful when there are different kinds of reasoning for the subnets that constitute a system. The function of the switch is to reset the conditions of the subnet and activate the corresponding reasoning process. The prototype developed in the present work deals only with dynamic activities (event-driven reasoning), therefore the switch place can be treated as a common place whose function is to reset the conditions of one specific subnet.

The prototype developed can do reasoning in complex nets, but there is only one level of nesting. A program can be developed to do reasoning with several levels of nested nets. The programmer should consider different aspects such as: every place or transition can be another net in a deeper level; the algorithms for nested nets could be developed in a recursive manner in order to use the same code for different levels of reasoning; every subnet should contain a place or transition as an initial alerter to start the process in that specific node when the algorithm reaches the subnet; the final results of a subnet should be sent to the upper level net to maintain the flow of information; the program could create subnets dynamically and attach them to the upper net as a single node; every subnet should have all the descriptions and characteristics of the nodes; in other words, every subnet should define its own places, transitions, marking vector, enabled vector, active vector, sequence queue, result vector, input and output functions, etc.

Under these conditions it is possible to develop a powerful tool to analyze complex systems which are not easy to represent in a single net.

CHAPTER VI

CONCLUSIONS AND FUTURE WORK

The initial design of the Petri net representation for a system is a process that involves precise knowledge of the system and usually should be carried out by more than one expert, especially when concurrent activities are involved. Sometimes designers must create places and transitions that are not obvious or clearly specified in the real system. One way to facilitate the initial design is to take small parts of the system and design the corresponding Petri net for each one independently; after that , it is easier to integrate all the modules in a higher level net.

In the future, a graph interface should be developed to facilitate the initial design and the modifications or upgrades of the system. With this interface the user would probably not have to code the net representation; thus the possibility of making mistakes will be reduced.

Although the software developed to analyze Petri nets so far tends to be application-dependent (due to the assumptions of developers and the general restrictions of the nets) the present tool can be used to analyze information systems by introducing modifications to the original design. The user or knowledge engineer can observe the behavior of the system by introducing or deleting nodes or connections, or by changing the transition functions. It is especially useful in analyzing control systems that include feedback data to tune the system or reach certain optimal conditions. Also, it could be useful in developing software and controlling the information flow among modules or nodes in a network or any information system.

The tool developed in the present work can be used to develop expert systems because the reasoning process based on Petri nets can traverse the correct sequence or sequences of activities to reach the final goal of the system.

With the philosophy of Petri nets a user can design complex or nested nets, where a place or transition is itself another net. The subnets can be connected using "switch places"; however, all connections should be specified in detail, because the main function of a switch place is to reset the initial conditions of the subnet that it controls and pass the information needed by the subnet. Therefore, it will be convenient if, in the future, an algorithm is implemented for analyzing nested nets with several levels; for this purpose the knowledge table is useful because any slice can be another knowledge table.

In addition to general rules of Petri nets, coloured Petri nets and Knowledge Tables, the present work adds some rules for representing knowledge. These rules are related with the data flow among nodes; in other words, these are specific rules for the messages and the tokens which are presented in chapter IV.

The model proposed as a combination of Petri nets, Knowledge tables, and Control tables is a flexible and modular technique because the representation is independent of the reasoning strategies; the elementary nets can be combined to build complex nets. The model can be applied to the specification of information systems and for the simulation and prototyping of asynchronous activities.

In the prototype the transition functions are fixed, so every time a transition is fired the tokens are moved according to specific transition functions. It would be interesting to consider the possibility of having several transitions functions for a specific transition and place; with this feature a system could be more dynamic because the movement of tokens and their colors would be used according to the results of the transition.

BIBLIOGRAPHY

1 Brady, N.C. THE NATURE AND PROPERTIES OF SOILS. Macmillan Publishing Co. Inc. 8th edition. New York. 1974.

2 Caudill, M. USING NEURAL NETS: REPRESENTING KNOWLEDGE. AI Expert. Vol.4, No.12. pp.34-41. December 1989.

3 Chang, S.K. and Ho, C.S. KNOWLEDGE TABLE AS A UNIFIED KNOWLEDGE REPRESENTATION FOR OFFICE AUTOMATION SYSTEM DESIGN. Office Knowledge Engineering, Newsletter of IEEE TC on Office Automation. Vol.3, No.1. pp.12-25. February 1989.

4 Charniak, E. and McDermott D. INTRODUCTION TO ARTIFICIAL INTELLIGENCE. Addison-Wesley Publishing Co. California. 1986.

5 Chen, S-M. Ke, J-S. and Chang J-F. KNOWLEDGE REPRESENTATION USING FUZZY PETRI NETS. IEEE Transactions on Knowledge and Data Engineering. Vol.2, No.3. pp. 311-319. September 1990.

6 Deng, P., Holsapple C. W. and Whinston A. B. A SKILL REFINEMENT LEARNING MODEL FOR RULE-BASED EXPERT SYSTEMS. IEEE Expert. Vol.5, No.2. pp. 15-27. April 1990.

7 Deng, Y. and Chang S-K. A G-NET MODEL FOR KNOWLEDGE REPRESENTATION AND REASONING. IEEE Transactions on Knowledge and Data Engineering. Vol.2, No.3. pp. 295-310. September 1990.

8 Deng, Y. and Chang S. A FRAMEWORK FOR THE MODELING AND PROTOTYPING OF DISTRIBUTED INFORMATION SYSTEMS. International Journal of Software Engineering and Knowledge Engineering. September 1991.

9 Deng, Y. Department of Computer Science, University of Pittsburgh, PA 15260. Personal communication via E-mail and telephone. 1991.

10 Fordyce, K. Jantzen, J. Sullivan, G. A. Sr. and Sullivan, G.A. Jr. REPRESENTING KNOWLEDGE WITH FUNCTIONS AND BOOLEAN ARRAYS. IBM Journal of Research and Development. Vol.33, No.6. pp. 627-645. November 1989.

11 Hunt, V.D. ARTIFICIAL INTELLIGENCE AND EXPERT SYSTEMS SOURCEBOOK. Chapman and Hall. New York. 1986.

12 Jacob, R.J.K. and Froscher J.N. A SOFTWARE ENGINEERING METHODOLOGY FOR RULE-BASED SYSTEMS. IEEE Transactions on Knowledge and Data Engineering. Vol.2. No.2 pp. 173-185. June 1990.

13 Jensen, K. COLOURED PETRI NETS. Lecture Notes in Computer Science. Vol.254. pp.249-299. 1987.

14 Kasturia, E. DiCesare, F. and Desrachers, A. REAL TIME CONTROL OF MULTILEVEL MANUFACTURING SYSTEMS USING COLORED PETRI NETS. Proceedings - 1988 IEEE International Conference on Robotics and Automation. pp. 1114-1119. April 1988.

15 Mayfield, B.E. Assistant Professor, Computing and Information Sciences, Oklahoma State University, Stillwater. 1991.

16 Murata, T. PETRI NETS: PROPERTIES, ANALYSIS AND APPLICATIONS. Proceedings of the IEEE. Vol.77. pp. 541-580. April 1989.

17 Obermeir, K.K. and Barron, J.J. TIME TO GET FIRED UP. Byte. Vol.14, No.8. pp. 217-224. August 1989.

18 Parsaye, K. ACQUIRING AND VERIFYING KNOWLEDGE AUTOMATICALLY. AI Expert. Vol.3, No.5. pp.48-63. May 1988

19 Peterson, J.L. PETRI NETS. Computing Surveys. Vol.9, No.3 pp.223-250. September 1977.

20 Ribaric, S. KNOWLEDGE REPRESENTATION SCHEME BASED ON PETRI NET THEORY. International Journal of Pattern Recognition and Artificial Intelligence. Vol.2, No.4. pp. 691-700. December 1988.

21 Roberts, L. ARE NEURAL NETS LIKE THE HUMAN BRAIN?. Science. Vol.243. pp.481-482. January 27, 1989.

22 Schildt H. C THE COMPLETE REFERENCE. Osborne McGraw-Hill. Berkeley, California. 1987.

23 Shapiro, S.C. Encyclopedia of Artificial Intelligence. Vol.1-2. John Wilwy and Sons. New York. 1987.

24 Simmons, A.B. and Chappel, S.G. ARTIFICIAL INTELLIGENCE - DEFINITION AND PRACTICE. IEEE Journal of Oceanic Engineering. Vol.13 No.2. pp. 14-41. April 1988.

25 Slagle, J.R. and Gardiner, D.A. KNOWLEDGE SPECIFICATION OF AN EXPERT SYSTEM. IEEE Expert. Vol. 5. No.4 pp.29-37. August 1990.

26 Smith, R. THE FACTS ON FILE DICTIONARY OF ARTIFICIAL INTELLIGENCE. Facts on File. New York. 1989.

27 Touretzky, D.S. and Pomerlau D.A. WHAT'S HIDDEN IN THE HIDDEN LAYERS?. Byte. Vol.14, No. 8. pp.227-233. August 1989.

28 Walters, J. and Nielsen, N.R. CRAFTING KNOWLEDGE-BASED SYSTEMS. John Wiley and Sons. New York. 1988.

29 Westerman, R.L. FACTORS AFFECTING SOIL ACIDITY. Solutions Magazine. May-June 1981.

30 Westerman, R.L. Head of Agronomy Department, Oklahoma State University, Stillwater, Oklahoma. 1991.

APPENDIXES

Appendixes A and B contain the description of the systems used in the examples and the trace or flow of reasoning process.

The description of the system is the same input file that contains the specifications of places and transitions, and the initial conditions.

The trace of the reasoning process basically shows: a) the contents of the execution queue after any important modification of the queue; b) the execution of the activities in the places with the corresponding results; the order and meaning of those results depend on the specific system being analyzed; c) the execution of the activities in the transitions with corresponding results.

If the user has designed the system as a cycle, the reasoning process asks if he/she wants to run another cycle after reaching the initial alerter which is the starting point of the cycle.

APPENDIX A

TRACE FOR EXAMPLE 1

```
 P E T R I   N E T   R E A S O N I N G
 -------------------------------------

DESCRIPTION OF THE SYSTEM AND INITIAL CONDITIONS
 (Input File)

 Name : Lime Recommendation for Acid Soils.

 Number of Places = 6
 read   a  0  LRREAD
 *TIN:    if1 if2
 *TOUT:    t3
 bsreal  a  0  lrbsreal
 *TIN:    t1
 *TOUT:    if1
 bsth  a  0  lrbsth
 *TIN:    t2
 *TOUT:    if2
 bsnec  a  0  lrbsnec
 *TIN:    if3 if4
 *TOUT:    t1 t2
 lime  a  0  lrlime
 *TIN:    t4
 *TOUT:    if3
 print  a  0  lrprint
 *TIN:    t3
 *TOUT:    t4 if4

 Number of Transitions =  8
 if1  0  lrif1
 *PIN:    (read 1 1)
 *POUT:    (bsreal 1 1)
 if2  0  lrif2
 *PIN:    (read 1 1)
 *POUT:    (bsth 1 1)
 t1  1
 *PIN:    (bsreal 1 1)
 *POUT:    (bsnec 1 1)
 t2  1
 *PIN:    (bsth 1 1)
 *POUT:    (bsnec 1 1)
 if3  0  lrif3
 *PIN:    (bsnec 1 1)
 *POUT:    (lime 1 1)
 if4  0  lrif4
 *PIN:    (bsnec 1 1)
 *POUT:    (print 1 1)
 t4  1
 *PIN:    (lime 1 1)
 *POUT:    (print 1 1)
 t3  1
 *PIN:    (print 1 1)
 *POUT:    (read 1 1)
```

TRACE OF REASONING PROCESS

```
Contents of Queue:
     print p

Contents of Queue:
     print p    read p

Place  read  was executed
 Results:  st c 80 4.0 30 .6

Transition to be fired: if1
 Results:  1 st 0 c 0 80 4.0 30 .6

Contents of Queue:
     if1 t

Transition to be fired: if2
 Results:  0 st 0 c 0 80 4.0 30 .6

Contents of Queue:
     if1 t

Contents of Queue:
     read p

Contents of Queue:
     read p    bsreal p

Place  bsreal  was executed
 Results:  st c 80 4.0 30 .6 31.80 r

Transition to be fired: t1
 Results:  st c 80 4.0 30 .6 31.80 r

Contents of Queue:
     t1 t

Contents of Queue:
     bsreal p

Contents of Queue:
     bsreal p    bsnec p

Place  bsnec  was executed
 Results:  st c 48.200001 4.0 30 .6 r

Transition to be fired: if3
 Results:  1 st c 48.200001 4.0 30 .6 r 0

Contents of Queue:
     if3 t
```

```
Transition to be fired: if4
 Results:  0 st c 48.200001 4.0 30 .6 r 0

Contents of Queue:
     if3 t

Contents of Queue:
     bsnec p

Contents of Queue:
     bsnec p    lime p

Place  lime  was executed
 Results:  st c 48.200001 4.0 30 .6 r 24100.000000

Transition to be fired: t4
 Results:  st c 48.200001 4.0 30 .6 r 24100.000000

Contents of Queue:
     t4 t

Contents of Queue:
     lime p

Contents of Queue:
     lime p    print p

Place  print  was executed
 Results:  st c 48.200001 4.0 30 .6 r 24100.000000

Transition to be fired: t3
 Results:  st c 48.200001 4.0 30 .6 r 24100.000000

Contents of Queue:
     t3 t

*** End of Reasoning ***
-------------------------------
```

APPENDIX B

TRACE FOR EXAMPLE 2

```
 P E T R I   N E T   R E A S O N I N G
 -------------------------------------

DESCRIPTION OF THE SYSTEM AND INITIAL CONDITIONS
 (Input File)

 Name : Mathematic Computations


 Number of Places = 12
 p1  a  0  p1a
 *TIN:   copya
 *TOUT:   read
 p2  a  0  p2b
 *TIN:   copyb
 *TOUT:   read
 p3  v  0
 *TIN:   add
 *TOUT:   copya
 p4  v  0
 *TIN:   subtr
 *TOUT:   copya
 p5  v  0
 *TIN:   add
 *TOUT:   copyb
 p6  v  0
 *TIN:   subtr
 *TOUT:   copyb
 p7  v  0
 *TIN:   divide
 *TOUT:   add
 p8  v  0
 *TIN:   if1 if2
 *TOUT:   subtr
 p9  v  0
 *TIN:   divide
 *TOUT:   if1
 p10  v  0
 *TIN:   true1
 *TOUT:   divide
 p11  v  0
 *TIN:   true2
 *TOUT:   if2
 start  s  1
 *TIN:   read
 *TOUT:   true1 true2

 Number of Transitions =  10
 read  0  read
 *PIN:
 *POUT:   (p1 1 1) (p2 1 1)
 copya  1
 *PIN:   (p1 1 1)
 *POUT:   (p3 1 1) (p4 1 1)
 copyb  1
 *PIN:   (p2 1 1)
 *POUT:   (p5 1 1) (p6 1 1)
 add  0  add
 *PIN:   (p3 1 1) (p5 1 1)
 *POUT:   (p7 1 1)
 subtr  0  subtr
 *PIN:   (p4 1 1) (p6 1 1)
 *POUT:   (p8 1 1)
 divide  0  divide
 *PIN:   (p7 1 1) (p9 1 1)
 *POUT:   (p10 1 1)
 if1  0  if1
 *PIN:   (p8 1 1)
 *POUT:   (p9 1 1)
 if2  0  if2
 *PIN:   (p8 1 1)
 *POUT:   (p11 1 1)
 true1  1
 *PIN:   (p10 1 1)
 *POUT:   (start 1 1)
 true2  1
 *PIN:   (p11 1 1)
 *POUT:   (start 1 1)


TRACE OF REASONING PROCESS


Place  start  was executed
 Results:

Transition to be fired: read
 Results:  1 123.45 122.0

Contents of Queue:
     read t

Contents of Queue:
     p1 p

Contents of Queue:
     p1 p    p2 p

Place  p1  was executed
 Results:  123.45

Transition to be fired: copya
 Results:  123.45

Contents of Queue:
     p2 p    copya t
```

```
Place  p2  was executed
 Results:  122.0

Transition to be fired: copyb
 Results:  122.0

Contents of Queue:
     copya t    copyb t

Contents of Queue:
     copyb t    p1 p

Contents of Queue:
     copyb t    p1 p    p3 p

Contents of Queue:
     copyb t    p1 p    p3 p    p4 p

Contents of Queue:
     p1 p    p3 p    p4 p    p2 p

Contents of Queue:
     p1 p    p3 p    p4 p    p2 p    p5 p

Contents of Queue:
     p1 p    p3 p    p4 p    p2 p    p5 p    p6 p

Place  p3  was executed
 Results:  123.45

Transition to be fired: add

Place  p4  was executed
 Results:  123.45

Transition to be fired: subtr

Place  p5  was executed
 Results:  122.0

Transition to be fired: add
 Results:  1 245.449997

Contents of Queue:
     p6 p    add t

Place  p6  was executed
 Results:  122.0

Transition to be fired: subtr
 Results:  1 1.449997

Contents of Queue:
     add t    subtr t

Contents of Queue:
     subtr t    p3 p

Contents of Queue:
     subtr t    p3 p    p5 p

Contents of Queue:
     subtr t    p3 p    p5 p    p7 p

Contents of Queue:
     p3 p    p5 p    p7 p    p4 p

Contents of Queue:
     p3 p    p5 p    p7 p    p4 p    p6 p

Contents of Queue:
     p3 p    p5 p    p7 p    p4 p    p6 p    p8 p

Place  p7  was executed
 Results:  245.449997

Transition to be fired: divide

Place  p8  was executed
 Results:  1.449997

Transition to be fired: if1
 Results:  1 1.449997

Contents of Queue:
     if1 t

Transition to be fired: if2
 Results:  0 1.449997

Contents of Queue:
     if1 t

Contents of Queue:
     p8 p

Contents of Queue:
     p8 p    p9 p

Place  p9  was executed
 Results:  1.449997

Transition to be fired: divide
 Results:  1 169.276215

Contents of Queue:
     divide t
```

```
Contents of Queue:
     p7 p

Contents of Queue:
     p7 p    p9 p

Contents of Queue:
     p7 p    p9 p    p10 p

Place  p10  was executed
 Results:  169.276215

Transition to be fired: true1
 Results:  169.276215

Contents of Queue:
     true1 t

Contents of Queue:
     p10 p

Contents of Queue:
     p10 p    start p

*** End of Reasoning ***
-------------------------------
```

VITA

RAFAEL ORTIZ

Candidate for the Degree of

Master of Science

Thesis: KNOWLEDGE REPRESENTATION USING PETRI NETS AND KNOWLEDGE TABLES

Major Field: Computer Science

Biographical:

Personal Data: Born in Jesus Maria, Santander, Colombia, March 15, 1955, the son of Rafael and Imelda.

Education: Graduated from "Externado Nacional Camilo Torres" High School, Bogota, Colombia in November 1973; received Ingeniero Electricista degree from "Universidad Nacional de Colombia", Bogota in December 1986; completed requirements for the Master of Science degree at Oklahoma State University in December, 1991.

Professional Experience: Computer programer and systems analyst, Department of Statistics, Colombian Institute of Agriculture, January, 1987, to the present.