STRUCTURING THE DEVELOPMENT OF PRODUCTION SYSTEMS

by

RODNEY D. ANDERSON

B. S., Kansas State University, 1987

M. S., Kansas State University, 1988

A MASTERS THESIS

submitted in partial fulfillment of the

requirements for the degree

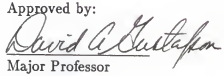
MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

Approved by:



Major Professor

Acknowledgements

I wish to thank first of all my parents who made it all possible with encouragement and support. I wish to thank Dr. David L. Gustafson for his dedication, inspiration, and insight. Our weekly meetings always produced new questions or ideas, prompting me to continue working. Thanks also to various students who supplied me with their work so that I could empirically judge mine. And finally thanks to the philosopher Francis Bacon. In his work, Advancement of Learning, he observed the very core of research :

"If a man will begin with certainties, he shall end in doubts; but if he will be content to begin with doubts, he shall end in certainties."

TABLE OF CONTENTS

CHAPTER ONE

Introduction

The advent of expert systems may trace its roots back to the early 1950's and the beginnings of artificial intelligence. John McCarthy first coined the phrase "artificial intelligence" as scientists attempted to embody human-like intelligence in computer programs. As opposed to conventional programming techniques, this new approach strove to capture human reasoning capabilities in general problem solving and to apply them to all problems. Eventually it was evident that a general problem solving program was not applicable to all types of problems and another approach was definitely needed. The early 70's found the answer. Instead of attempting to capture all knowledge for all domains, researchers realized that even human experts were limited to a narrow domain of expertise. Therefore, program's domains should be limited. In this manner, the task of modelling a human expert's thought processes would be simpler. This was the beginning of expert systems development.

Still a relatively new technology, expert systems have proven themselves capable of meeting the challenge of a new frontier in computer science. But, in order to further advance the field, we now need to start improving the development process of expert systems. This paper will present a diagrammatical approach to the construction of expert systems that leads to a more structured knowledge base.

To begin with, the paper gives a brief overview of expert systems including definitions, distinctive features, compositional makeup, and contrasts with conventional programs. Then the paper discusses current development techniques of expert systems including rapid prototyping, shells, and iterative development. Next, the paper discusses the application of software engineering techniques and practices to expert systems. A proposal for a diagrammatical approach to development is then presented.

CHAPTER TWO

Expert System Overview

"An expert system is regarded as the embodiment within a computer of a knowledge-based component, from an expert skill, in such a form that the system can offer intelligent advice or make an intelligent decision about a processing function [1]." That is, an expert system is a system which stores a human expert's knowledge concerning a narrow problem domain and applies that knowledge towards the solution of problems in that domain.

2.1 Distinctive Characteristics of an Expert System [1]

1) Expertise limited to a specific domain
2) Separation of the expert knowledge and the use of that knowledge
3) Explanatory facilities of its reasoning
4) Ability to reason with uncertain or judgmental data using fuzzy logic, bayesian methods, etc.
5) Typically expressed in heuristic rule form

2.2 Contrasts to Conventional Programming

These characteristics contrast expert systems with conventional programs. Whereas, the data and algorithms form a conventional programming solution to problems, expert system's solutions are composed of knowledge and inference techniques to apply that knowledge. Further, conventional programs are built from an ordered sequence of unambiguous statements (an algorithm) where the control of statement

execution has to be precisely defined using sequential, conditional, or iterative structures. However, expert system programs are usually an unordered collection of rules describing situations and corresponding actions to take if that situation occurs. Control is not specified in the program. Finally, conventional programs are designed to produce correct and efficient results. Expert systems are modelled to follow a human's thought processes and therefore expert systems may be erroneous or may not produce the optimal solution.

2.3 Specialization of the Problem Domain

In the course of early expert system's development, researchers noticed the logical separation of an expert system into two parts. The first part, the inference mechanisms, contained generalized heuristics that were applicable to different systems. The second part, the knowledge base, contained problem specific information needed for the particular domain of interest where the problem solving power resides in the possession of the knowledge derived from the expert. The general goal of the expert system's implementation then became construction of the knowledge base, using pre-existing inference procedures. This effectively narrowed the problem to domain specific information. However, further refinement of the domain may be needed to ensure a feasible system. In general, a human expert has expert knowledge or reasoning capabilities in a limited scope or area. Similarly, an expert system can only be effective in a narrow problem area.

For these types of systems, there are suitable and unsuitable problem domains.

Suitable Problem Domain Properties: [2]

1) Domain experts exist
2) Experts can articulate their methods
3) Experts agree on solutions
4) Task does not require common sense
5) Task requires only cognitive skills
6) Task is not poorly understood
7) Task is not too difficult

Unsuitable Problem Domains: [3]

1) Efficient algorithmic solutions exist
2) Tasks are sequential in nature
3) Tasks require precise flow of control
4) Extensive numerical approximations or calculations are required.

The above are merely guidelines and not necessarily strict rules to follow. There are in fact uses for expert systems that fall outside of these guidelines. So, any problem must be analyzed with regard to cost and benefit tradeoffs.

2.4 Building an Expert System

Once the problem area is defined, building an expert system is a process which begins by extracting the expert's knowledge. A knowledge engineer is responsible for this task. To build successful expert systems, we need human experts that can articulate their methods. Through interviews with the expert and/or observations of

his/her problem-solving abilities, the knowledge engineer must obtain a set of guidelines describing the expert's process of analyzing and solving the problem.

Most often, human experts can formalize their process of solution by the use of heuristics of the form if-then. If some condition is met, then they perform certain actions. Hayes-Roth has described four distinctive features of rules that follow [4].

1) they define a parallel decomposition of state transition behavior thereby inducing a parallel decomposition of overall system state that simplifies auditing and explanation.
2) they can simulate deduction and reasoning by expressing logical relationships and definitional equivalences.
3) they can simulate subjective perception by relating signal data to higher level pattern classes.
4) they can simulate subjective decision making by using conditional rules to express heuristics.

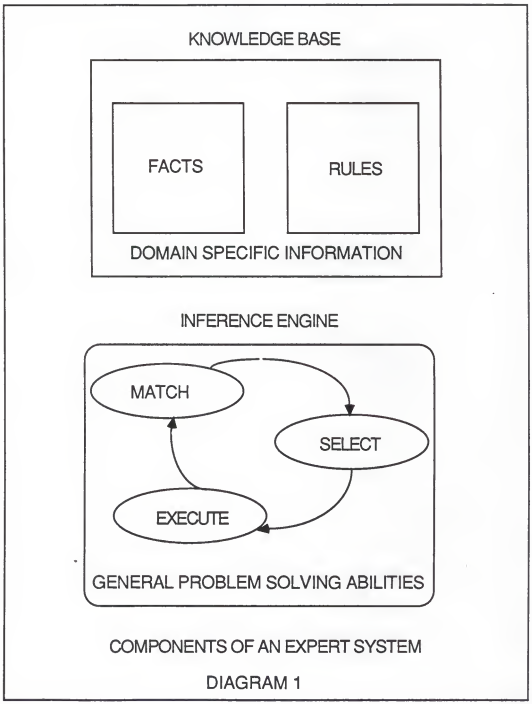
A collection of these rules should detail the solution process. New heuristics will be assimilated into the set of rules as the knowledge increases. One of the concerns at this point should be the organization of the rules into logical units. We must insure that the assimilation process does not fragment the system into an unorganized collection of rules where side-effects can flourish. As realized in software engineering, the decomposition of a program into modules that have high cohesion and low coupling greatly reduces the possibility of side-effects [5]. Without this structuring, several problems can exist including inefficient operation, inaccurate operation due to side-effects, and loss of maintainability.

Development of the physical system can now proceed. At present the most prevalent techniques are rapid prototyping [6] and reusability via shells. (The next chapter will define and analyze these two techniques.) However, these techniques are not conducive to structured design methodologies and often lead to incomplete, inaccurate systems.

2.5 Architecture of Rule-based Expert Systems

The basic architecture of rule-based expert systems (production systems) includes 3 parts: a set of facts, a set of rules, and an inference engine (See diagram 1). The facts are the knowledge components that are initially known or inferred throughout the process. The rules are problem specific conditional actions which resemble if-then statements. If some set of conditions is met, then some action is taken. The possible actions that can be taken are the modification, deletion, or addition of facts to the present fact memory. Together, these two components, the facts and the rules, are collectively known as the knowledge base. The third component is the inference engine which executes the rules. But before execution of a rule occurs, the inference engine must decide which rule from the rule set to fire.

To describe the inference engine's activities in more detail, we can view it as a three step process.



COMPONENTS OF AN EXPERT SYSTEM

DIAGRAM 1

Inference engine cycle

1) match step
2) selection step
3) execution step

In the first step of the process, the engine examines all of the rules satisfied by the current facts and groups them together into what is known as the conflict set. The conflict set consists of rules that have potential for execution. The second step of the process takes the conflict set and selects one of the rules to be executed based upon a conflict resolution strategy. The third step, performs the actions specified by the rule which can modify the current fact memory. Due to the change in the fact memory, a different conflict set will most likely occur for the next cycle of the process. Thus, control in an expert system is not based upon any static control structures as in conventional programming. It is driven by the facts and their modifications.

Once we have built the conflict set we need a strategy to select one rule to execute. A conflict resolution strategy utilizes a priority scheme to determine which rule from the conflict set to select. Several methods are utilized in different expert system languages. The following example is from the YAPS programming language which examines the age of the facts as its basis of selection [7].

YAPS Conflict Resolution Strategy

1) If a rule has already fired with a given set of facts, it will not fire again unless one of those facts is refreshed.

2) For each rule in the conflict set, sort the ages of the facts into a list in descending order.

3) Compare the first ages in the list for each rule and select the one with the smallest age (the most recent fact). If a tie occurs, compare successive ages in the lists. If a tie still exists and one list is longer than the others, select that rule. Ties that remain unresolved result in a random selection of a rule.

The rule is then executed, modifying the fact knowledge base. Continuing the inference engine cycle, a new conflict set is built from the match step and a new rule is selected. This process continues until instructed to halt or there are no rules present in the conflict set.

The two most common algorithms for inference engines are forward and backward chaining. "In a forward chaining system, a rule is triggered when changes in working memory data produce a situation that matches its antecedent component [4]." Thus, every rule whose condition is met by the current facts is executed. This can be very inefficient in that every rule of the system must have its conditions evaluated even if its actions are not relevant to the current line of reasoning. "In a backward chaining system, the rule based system begins with a goal and successively examines any rules with matching consequent components [4]." That is to say, it will

concentrate the selection process on rules relevant to the current line of reasoning. The efficiency here is that rules that will not contribute to this end are not executed. An example of these two types of algorithms follow.

Assume the current facts are:

Red Yellow Blue Black White

Assume the current rules are:

1) if Yellow and Blue then Green
2) if Black and White then Grey
3) if Green and Red then Orange

Forward chaining would initially place rules 1 and 2 into the conflict set. The select step would select rule 1 or 2 based on the conflict resolution strategy. Let us assume rule 1 is executed. This puts a new fact into the fact memory that is required by rule 3. Therefore, the next cycle will place rules 2 and 3 into the conflict set and execute one of them. Finally, only one rule is left in the conflict set, selected, and executed. Thus, forward chaining would result in execution of all three rules and a final set of facts of:

Red Yellow Blue Black White Green Orange Grey

Backward chaining requires a goal that we are trying to achieve. For purposes of example let us assume we want to establish the fact Orange. Rule 3 establishes the goal Orange for us, but only if Green and Red are current facts. Red is already a

fact. To achieve Green, we need to execute rule 1 using the current facts Yellow and Blue. Therefore, only rules 1 and 3 are executed resulting in a final set of facts of:

Red Yellow Blue Black White Green Orange

CHAPTER THREE

Analysis of Current Development Methods

Several different development approaches exist presently. However, they all are rather ad hoc methods in which there is no set structuring of events in the process. And often these events overlap throughout the development. Generically, development consists of five steps [8].

Five steps in the development process

1) identification of the problem
2) logical design
3) requirements design
4) implementation
5) testing

Identification of the problem specifies the important goals of the system. Logical design determines strategies to follow in solving the problem. Requirements design formalizes the system's expectations between the end-user and the developer. Implementation develops a workable program that may include several prototypes. Testing activities verify that the product works and can be relied on.

Although this generic classification exists, the steps are highly interdependent and overlapping. Arguments prevail that expert system development cannot be compared to conventional program development and that an iterative process for defining the requirements is necessary. With this in mind, several developers insist that rapid

prototyping is an effective way to establish development. This entails an iterative process whereby goals constantly shift, designs are revised, and several attempts at implementation are performed. Others insist that expert system tools such as shells are the best way to go, eliminating the knowledge engineer's job and relying on domain experts to supply the knowledge interactively. The next two sections will discuss each of these methods in more detail.

3.1 Rapid Prototyping

Rapid prototyping is a technique whereby a knowledge engineer proceeds to implement part of the system when exact requirements for the whole system are not yet established. "Recognizing this, one should build a prototype system fully expecting to throw away virtually all this code and start again [8]." Therefore the process of development exists in four stages: problem determination, initial prototype, expanded prototype, and delivered system [9]. The problem determination phase consists of realization of the feasibility of an expert system to solve a problem. Some form of functional specification occurs, but again it may be incomplete. The initial prototype phase is viewed as a quick method to prove the feasibility of the system and to solidify the requirements. At this stage, the initial prototype may be expanded iteratively to encompass the full scope needed and requires extensive interaction with the experts. Finally, the delivery system results by optimizing the prototype and refining the user interfaces.

It is argued that with the new technology of expert systems, user's attitude towards the new field is one of unbelief. Therefore, prototypes serve as the vehicle driving user interest to the point of wanting more. This is simply a public relations tool. However, knowledge engineers claim that prototypes give a better feel for the requirements of the system when vague objectives are given them. With a partial, up-and-running system, engineers can get more definitive interaction with the expert indicating what is right and what is wrong.

Yet, iterative development with these prototypes may lead to a knowledge base which is fragmented, where gaps in knowledge exist, and which may approach unmanageable size. At this juncture, the system efficiency is down and may even be unreliable, demanding redesign and reimplementation from the start [10]. While rapid prototyping has its place for experimentation with unusual new problems, it cannot effectively be utilized for development of large applications. As we have learned in conventional programming practices, applying software engineering techniques produces consistently better results. To summarize, when applications remain unclear between the expert and the knowledge engineer, then rapid prototyping may be a valid option to obtain physical results that can be critiqued. However, when specifications are clear and knowledge engineers gain maturity in approaching development, feasibility is not the issue and concerns shift to cost effectiveness [10].

3.2 Shells

Another method of development concerns the application of an expert system shell with built-in tools for acquiring and representing the knowledge. "An expert-system shell is an environment designed to support applications of a very similar nature and represents an intermediate point between specific applications and general-purpose knowledge engineering environments [8]." Since many tasks share common frameworks of solution, shells may provide the correct environment for expert system development for a new problem using a solution that worked for a previous problem. If so, work can be spent making the existing solution more efficient rather then spending time in development. Or, if a user is inexperienced in expert system development, this may provide an approach for development.

Many companies now market expert system shells as tools for non-technical users. However, these same companies would have people believe that the tools themselves are expert systems when in fact they are not. Additionally, we must remember that producers often overstate the applicability of their products. Not all expert systems can be accomplished with one shell. Again this reverts to original AI approaches to a general problem solver. Whereas tools such as shells can be extremely useful in the production of similar products (such as the Emycin shell) or beneficial for rapid prototype attempts, they are not panaceas. Some other restrictions with shells are 1) they usually are not built to interact with other software products, 2) they typically

use backward chaining techniques that may be unsuitable for some problems, 3) they may only have one knowledge representation technique limiting the knowledge to that structure, and 4) they do not address the problem of structuring rules in a logically consistent manner.

3.3 Conclusions

"Experimental studies demonstrate that the methodology is more useful than its accompanying tools [11]." While these methods have some practical usage in the new field of expert systems, they are not conducive to structured design methodologies. We need a structured approach to follow so that large scale development is not hampered by bad design techniques.

CHAPTER FOUR

Why Software Engineering Practices would be Beneficial to Expert Systems

Software engineering has brought programming-in-the-large considerable benefits such as cost effectiveness of team management, more consistently maintainable systems, and more reliable systems. It took computer scientists years of research to develop and validate such methodologies for procedural programming languages. However, expert systems are still in the infancy stage and we have not yet developed such techniques for them. Due to the contrasts between the two paradigms, we cannot directly apply procedural programming methodologies to expert systems. Therefore, we must proceed with attempts to develop an expert system development methodology so that the same benefits achieved in conventional programming can be realized in this new area as well.

4.1 Problems Impeding Expert System Solutions

"Developers often cite initial knowledge acquisition and large knowledge base maintenance as problems impeding expert system solutions [12]." The first step in building an expert system is acquiring the expertise required from a human expert. The knowledge engineer must extract the solution or analysis process that the human uses in solving the problem. But, the knowledge engineer must be careful not to misinterpret the process which might result in an incorrect system. A methodology of

communication is needed to develop a clear, concise and correct understanding of the process - no easy task. This is analogous to conventional program's requirements analysis phase.

Since expert systems are still in the infancy stage, we have not approached the serious question of maintenance once they are in operation. New requirements may surface that will require updating the knowledge base or changing rule patterns. Since knowledge is acquired incrementally, the system should be allowed to grow. Proper requirements and documentation throughout the development process will help the maintainer determine the effect of new rules added to the knowledge base.

4.2 An Empirical Study of a System Development

In a study for the Imperial Cancer Research Fund, Alvey, Myers, and Greaves investigated augmentation of a expert system used in the diagnosis of leukemia [13]. The system was prototyped first using the Emycin shell. Then it was developed in Prolog. The authors concluded four ideas essential for development of large scale expert systems.

First, an expert system is not simply a collection of rules. Rules can be grouped together when sharing a common focus and these must be consistent with one another. Therefore, revisions must be concerned with rule groups and not individual rules. This would correlate to modules in a conventional style program, where the developers must insure the correct functioning of that module independently of

others. Second, rule groups' design must insure that no gaps in the knowledge exist. This is similar to conventional programs' integration testing procedures. Third, rules that make conclusions based on the inability to conclude anything else, should be avoided. "If there are any errors in the rules for proving the item, the system is liable to give the wrong answer, but more importantly, the erroneous rules will escape detection [13]." Relating to conventional programs, this problem is analogous to faulty logic. Fourth, extracting the expertise from the human expert is best done by thoroughly defining the domain and then defining the rules concerning the domain. If the knowledge is obtained bit by bit, then "a poorly coordinated collection of rules and a poor representation of domain [13]" is likely to occur. Therefore, we need a communication methodology between the knowledge engineer and the expert so that extracting the expertise is accomplished in a thorough manner. Fifth, it is fairly common to misinterpret the expert during the interview process. This may cause the knowledge engineer to proceed with wrong conceptions. Therefore, a review system is needed to verify rules with the expert.

A similar conclusion was reached in software engineering with peer reviews of design before implementation proceeded. Generally, the requirements and design are shown as dataflow diagrams and/or hierarchical diagrams as a means to show the logical structure of the system. In database design, an entity-relationship diagram is produced so that the designer can verify operations with the customer before proceed-

ing with the physical design. Both of these examples use diagrammatical approaches. If these proven methods have worked in other areas, they warrant research in the expert system field.

Basically, diagrams can help us identify the structure of the rules during the knowledge acquisition phase and can help us maintain this structure throughout the lifecycle of an expert system. The proposed methodology in this paper will utilize a diagrammatical view of the expert system to be verified with the expert before implementation proceeds. The next chapter introduces this diagrammatical approach for depicting expert systems.

CHAPTER FIVE

Diagrammatical Representation

From Hayes-Roth's set of features [4], we can visualize the rules in an expert system as forming sets of rules whereby each set corresponds to some given activity. For the system to fire rules in this set, the system must have at least one condition that is common for all the rules in the set. If we think of this condition as a activity variable, then Hayes-Roth's first distinction of a rule based system is simply the progression from activity to activity as the system moves towards a solution via the changing activity variable. This is the basis for the activity diagram that follows. The activity diagram shows the total possible activities that can occur in the system and shows the possible flow of control from one activity to another. At the start of the system, it must be in a start or beginning activity (1.3) that initializes the system to begin processing. Eventually, the system will terminate execution and produce some results. There may be many of these halt activities (1.4) that exist for various conclusions reached or error conditions. Also, there are no restrictions that limit the effective operating activity to one. This allows concurrency (1.2.2).

5.1 Activity Diagram Formalization

The activity diagram is mathematically a four-tuple (S, T, s, f) such that:

S is a set of activities

T is a set of transitions from activity to activity

s is the start activity for the system

f is the set of halt activities for the system

where s is an element of S and f is a subset of S.

The activity diagram is a labelled directed graph representing the various activities and transitions in a production system. Nodes are the activities and directed arcs are the transitions from one activity to another.

Activity Diagram :

1.1 Each activity is drawn as either an oval or a rectangle. An oval is used if the newly instantiated activity makes a direct transition to another activity. A rectangle is used if the newly instantiated activity may remain in the current activity before a transition to another activity. Hereafter the activity entity will refer to the activity drawn as either an oval or a rectangle.

1.1.1 The name of the activity or condition for that activity is written inside the entity.

1.1.2 Annotating the entity is the number of rules utilizing this activity if known.

1.2 If a transition exists from activity i to activity j, it is drawn with a directed arc.

1.2.1 If the transition occurs by disabling the current activity and enabling the new activity, the arc is drawn with a solid line.

1.2.2 If the transition occurs without disabling the current activity, the arc is drawn with a dashed line.

1.3 The start activity is distinguished by START as the name of the activity (1.1.1) or by the word 'start' adjacent to the entity on the outside.

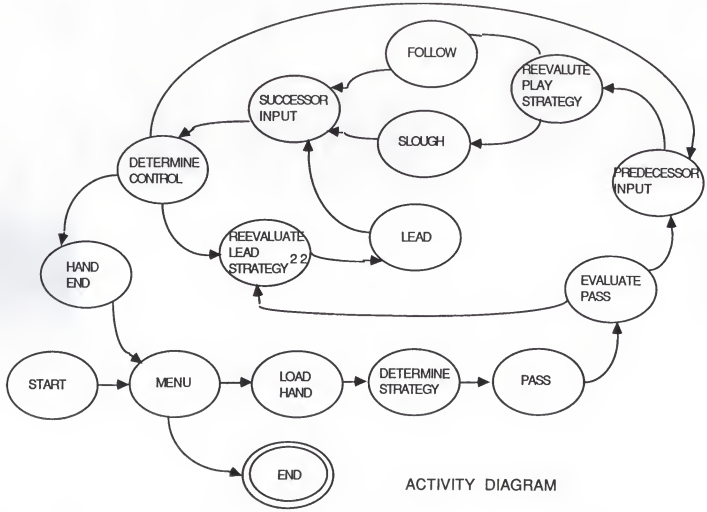
1.4 The termination activities are distinguished by double entities. Either double ovals or double rectangles.

5.2 Activity Diagram Example

For an example the problem selected was the card game hearts. In hearts, the objective is to have the lowest score. When the game ends, the player with the lowest point total wins. Points are accumulated each round by assigning one point for each heart that a player has taken and thirteen points for the queen of spades. Therefore, the objective is not to take the queen of spades or any hearts if at all possible. One exception exists. If a player can manage to take every heart and the queen, then that player is assigned no points for the round and every other player is assigned twenty-six points. A complete set of rules for hearts may be found in Hoyle's famous book [14].

Diagram 2 was created as the activity diagram for the system. At this point, it may not be clearly differentiable as to which activities utilize direct or indirect transitions. Therefore, ovals are used for all activities. As design continues, the difference should become apparent and taken into account in revisions of the activity diagram. One further note, activities in the initial activity diagram may be decomposed into several activities during the design process on large systems. This can be accomplished by abstracting the process into subparts for the initial diagram and refining individual parts as needed. However, such refinement should not be confused with the rule decomposition that follows in the rule diagram.

From the activity diagram and using Hayes-Roth's fourth feature, the process of



ACTIVITY DIAGRAM

DIAGRAM 2

decomposing the activities into individual rules can be achieved with the rule diagram. The rule diagram will show the possible interaction between rules existing in the system. Progression from one rule to another is decided by the inference engine's conflict resolution policy, but can be directly influenced by the programmer. If the programmer has in mind a direct ordering of the firing from one rule to another, an explicit flow will exist between those two rules (2.2.1). This dictates changing the activity variable from the one required for the first rule to that required by the second rule. On the other hand, the programmer may not want to sequence the control but will leave that to the inference engine, relying on its conflict resolution strategy if several rules utilize the same activity variable. This is an indirect flow between rules (2.2.2).

5.3 Rule Diagram Formalization

The Rule diagram is mathematically a five-tuple (R, C, T, s, f) such that:

R is a set of rules

C is a set of conditions

T is the transition function mapping R x C to R.

s is the start rule for the system

f is the set of halt rules for the system

where s is an element of R and f is a subset of R

The rule diagram is a labelled directed graph representing the various rules and transitions in a production system. Nodes are the rules and directed arcs are the transitions from one rule to another.

Rule Diagram :

2.1 Each rule is drawn as an oval.

2.1.1 The name of the rule is written inside the oval.

2.1.2 Annotating the oval is the number of conditions required for this rule to fire if known.

2.2 Flows from rule i to rule j are drawn with a uni-directional arc.

2.2.1 If an explicit flow exists, the arc is drawn with a solid line. An explicit flow from rule i to rule j exists if rule i adds a condition required by rule j.

2.2.1.1 If the rule is conditional on a test, then the arc is labelled with a name reflecting the test.

2.2.2 If an implicit flow exists, the arc is drawn with a broken line. An implicit flow from rule i to rule j exists if rule i and j require the same state value and rule j has the next highest priority for firing after rule i.

2.3 The start rule is distinguished by START as the name of the rule (1.1.1) or by the word 'start' adjacent to the oval on the outside.

2.4 The termination rules are distinguished by double ovals.

5.4 Rule Diagram Example

Using the example, we will decompose the "re-evaluate lead strategy" activity into three rule diagrams (diagrams 3a, 3b, and 3c). Each entity represents a rule. The conditions for the rules are labelled on the arcs. Decomposition of the activity into rules can proceed by focusing attention just on the concept at hand. This

method eliminates the need to constantly keep the whole concept in mind. This particular activity was decomposed into 22 rules. In order to clearly show these rules, the diagram is split into three parts. Therefore, connections are needed between rules in different diagrams. Connections are facilitated by the use of a numbering system, where arcs are aligned via matching numbers across diagrams.

As various activities are decomposed into rule diagrams, we may notice the same rule appearing in more than one activity. Just as conventional programming may use the same function in another part of the program, rules may be utilized more than once. Therefore, a complete rule diagram would show this rule once with the different transitions connected to it. But, separate rule diagrams could show the rule in the context of each activity decomposition. Above all, the concept behind the activity decomposition is to structure activities into subparts and then to build the subparts from rule sets.

5.5 Clarification of Activity Diagram Subpart 1.2

Subpart 1.2 of the activity diagram formalization nullifies the typical state transition diagram restriction limiting the effective operating state to one. State transition diagrams functionally map a state i to a state j based on the input received while in state i. This mapping is a one-to-one function. Although there may be many possible transitions to different states, only one is allowed to be taken terminating the current state as it leaves.

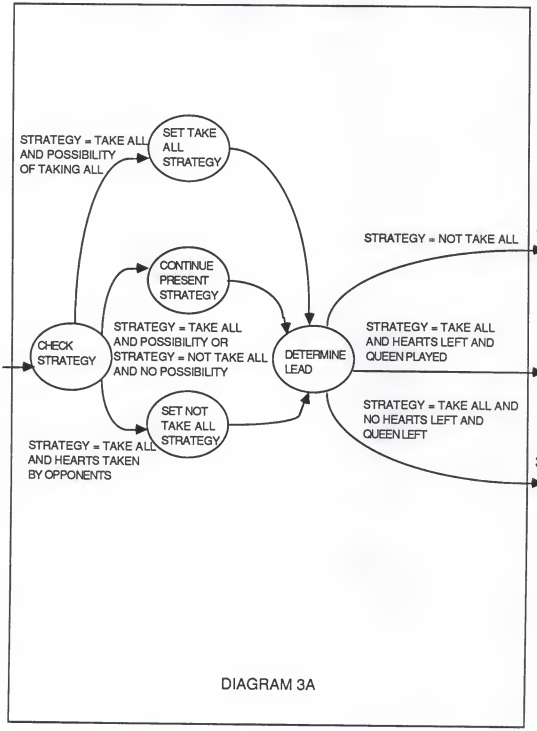

STRATEGY = TAKE ALL AND POSSIBILITY OF TAKING ALL
SET TAKE ALL STRATEGY
CONTINUE PRESENT STRATEGY
CHECK STRATEGY
STRATEGY = TAKE ALL AND POSSIBILITY OR STRATEGY = NOT TAKE ALL AND NO POSSIBILITY
SET NOT TAKE ALL STRATEGY
STRATEGY = TAKE ALL AND HEARTS TAKEN BY OPPONENTS
DETERMINE LEAD
STRATEGY = NOT TAKE ALL
1
STRATEGY = TAKE ALL AND HEARTS LEFT AND QUEEN PLAYED
2
STRATEGY = TAKE ALL AND NO HEARTS LEFT AND QUEEN LEFT
3

DIAGRAM 3A

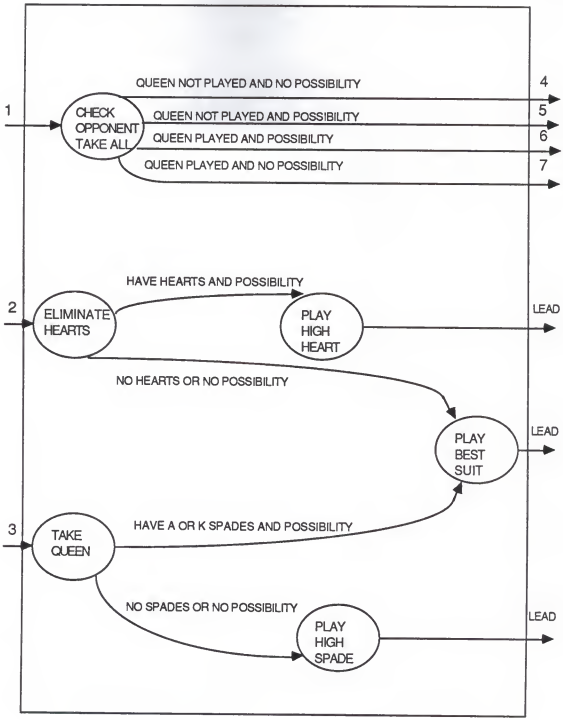

1
CHECK OPPONENT TAKE ALL
QUEEN NOT PLAYED AND NO POSSIBILITY
4
QUEEN NOT PLAYED AND POSSIBILITY
5
QUEEN PLAYED AND POSSIBILITY
6
QUEEN PLAYED AND NO POSSIBILITY
7
2
ELIMINATE HEARTS
HAVE HEARTS AND POSSIBILITY
PLAY HIGH HEART
LEAD
NO HEARTS OR NO POSSIBILITY
PLAY BEST SUIT
LEAD
3
TAKE QUEEN
HAVE A OR K SPADES AND POSSIBILITY
NO SPADES OR NO POSSIBILITY
PLAY HIGH SPADE
LEAD

DIAGRAM 3B

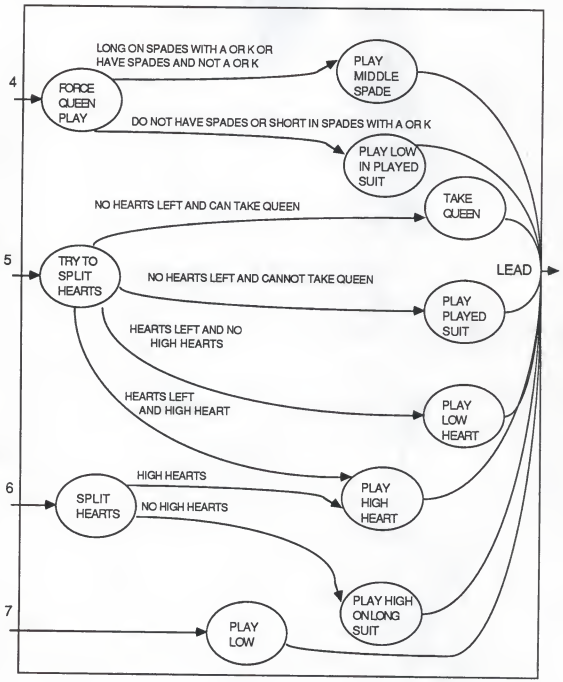

4
FORCE QUEEN PLAY
LONG ON SPADES WITH A OR K OR HAVE SPADES AND NOT A OR K
PLAY MIDDLE SPADE
DO NOT HAVE SPADES OR SHORT IN SPADES WITH A OR K
PLAY LOW IN PLAYED SUIT
NO HEARTS LEFT AND CAN TAKE QUEEN
TAKE QUEEN
5
TRY TO SPLIT HEARTS
NO HEARTS LEFT AND CANNOT TAKE QUEEN
PLAY PLAYED SUIT
HEARTS LEFT AND NO HIGH HEARTS
HEARTS LEFT AND HIGH HEART
PLAY LOW HEART
LEAD
6
SPLIT HEARTS
HIGH HEARTS
NO HIGH HEARTS
PLAY HIGH HEART
7
PLAY LOW
PLAY HIGH ON LONG SUIT

DIAGRAM 3C

However, there may exist times in expert systems where we wish to leave the current activity without disabling it. This can occur in a production system where activity i makes a transition to activity j without removing i's activity variable. Therefore, there are effectively two live activities in the system. Propagation of this effect will result in multiple active activities in the system. This may be beneficial if we want to return to a previous activity after performing another activity in the transition path. For example, suppose we have a system designed to detect and correct flow variations in a series of interconnected pipes. The monitoring activity may discover a flow interruption occurring. It therefore implements the tracker activity to pinpoint the problem and the correction activity to fix the problem. If the flow interruption ceases during the tracker activity, the tracker activity should be discontinued with control returning to the monitor.

Another benefit from this approach is the allowable introduction of concurrency. Expert systems can achieve great benefits from concurrency in allowing multiple activities to occur simultaneously. Such applications are needed in space technology and process monitoring. This diagrammatical approach can represent such systems.

5.6 Development of Diagrammatical Approach

This diagrammatical approach was developed by examining several systems and attempting to graphically depict the systems. The activity transition diagram seemed to best represent the components of the system, although other graphical approaches

were examined. For completeness, this section includes several of the systems diagrammed with a discussion of the structure of each system. However, as the systems are student or faculty projects, they will be labelled discretely for purposes of anonymity. Activity diagrams will be indicated by the suffix A after the diagram number. Rule diagrams will be indicated by the suffix B after the diagram number. For clarity in the connection of arcs, one node which branches to many other nodes will be drawn with a single line exiting the node eventually breaking into several arrows. This indicates several arcs from the one node to each of the other nodes.

Diagrams 4A and 4B represent a system that has not yet been completed and has difficulty terminating in a large state-space search. The system consists of 11 activities and 24 rules. Activities A through D set up the menu system and initialization activities. The inference process then begins at activity E. From here activities J, I, and F allow a network of transitions either returning to the menu on a successful completion or back to each other via activity G. From diagram 4B, we can see the decomposition of activities J, I, and F into 5 rules each. However, these rules contain a large number of conditions for that rule to fire. A more suitable approach would be to structure the network transitions better and to limit rules to a smaller number of conditions. If a rule's conditions exceeds some limit, the activity from which the rule was formed probably could have been further refined into subactivities requiring more rules, yet fewer conditions on those rules.

Diagrams 5A and 5B represent a component of a large system consisting of 16 activities and 24 rules. This system was included because of several unique discussion points. First, it allows iteration to continue in a structured way through a network of decisions as opposed to the previous example. Second, it shows the possibility of one rule utilized in different activities. For example, diagram 5B's rule R can be fired from three different activities. Third, the rules were designed using lisp conditionals in the consequent section of the rules to set up its transition to another rule. Therefore, most of the rules in the decision structure require only one condition, the activity variable. If the system was accomplished using the consequent section to set only the activity variable, a difference would be noticeable in the diagrams with rules showing more conditions. However, it solidifies the applicability of the diagrammatical method to variations of the rule-based approach.

A quantitative look at these two examples can use the arc to node ratio to measure the complexity of the system. A further refinement of this could include only the main part of the process as opposed to including the initial menu set-up. Diagram 4A's main section consists of activities E through J. Diagram 4B's main section consists of rules E through X. Diagram 5A's main section consists of activities C through P. Diagram 5B's main section consists of rules A through J, N, and P through S. The following table indicates the results of these two evaluations.

	All parts			Main Section		
Diagram	nodes	arcs	ratio	nodes	arcs	ratio
4A	11	19	1.7	6	11	1.8
4B	24	53	2.2	19	47	2.5
5A	16	27	1.7	14	24	1.7
5B	19	28	1.5	15	21	1.4

From this, diagram 5 has a lower ratio in the rule diagram which becomes more pronounced when evaluating the main section of the system. This indicates a cleaner design than diagram 4. A feasible argument at this point might indicate that diagram 4 was a more complex process than diagram 5, therefore it should have a higher ratio. However, this was not the case. In fact, diagram 5 represents a very complex task that was handled in a clean, well structured manner. Diagram 4 represents a conceptually easier task. Therefore, complexity of the process is not an issue. Diagram 5's program works, while diagram 4's program is not functioning. Structure of the process is the main concern. The diagrams can help us evaluate this structuring of the process.

Although the development of the diagrammatical method initially was formulated from examining existing systems, the previous discussion has substantiated its usefulness to evaluate system designs with regard to their structural complexity. The next step to ensure viability is to apply the approach to a development project. The main benefit in so doing should be the ability of using the diagrams to interact with the expert in structuring the knowledge base. The next chapter proposes a

development methodology for production systems using the diagrammatical method.

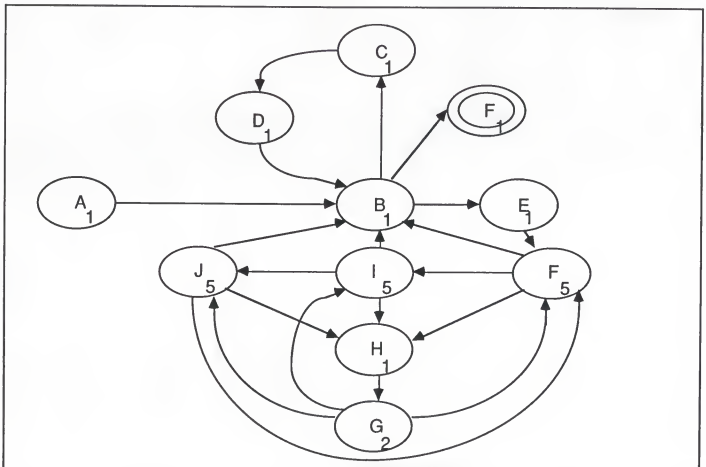


DIAGRAM 4A

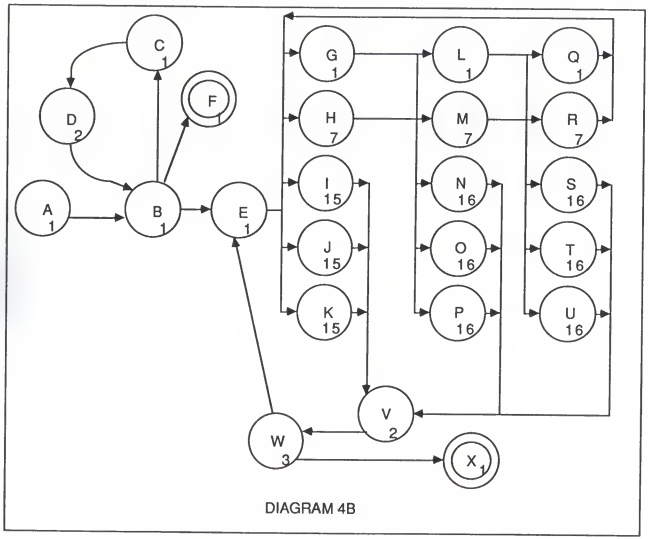

A
1
B
1
C
1
D
2
E
1
F
1
G
1
H
7
I
15
J
15
K
15
L
1
M
7
N
16
O
16
P
16
Q
1
R
7
S
16
T
16
U
16
V
2
W
3
X
1

DIAGRAM 4B

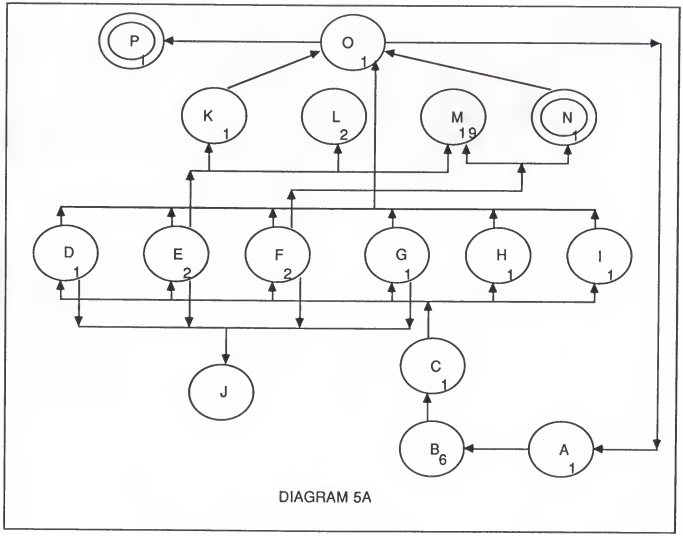


DIAGRAM 5A

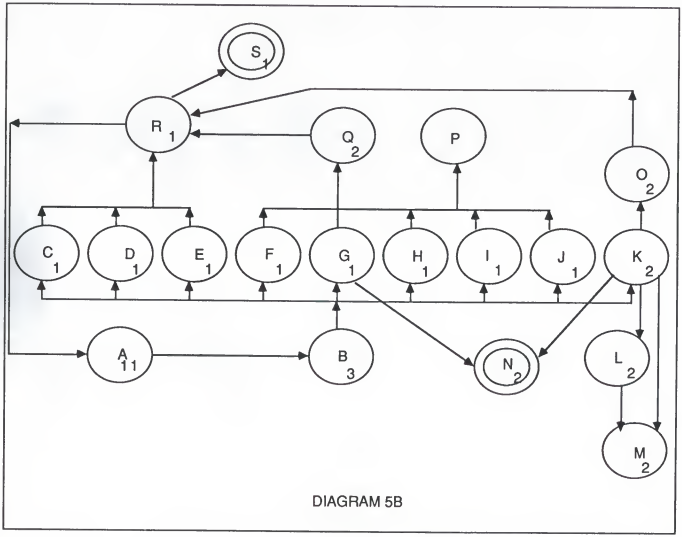


DIAGRAM 5B

CHAPTER SIX

A Development Methodology for Production Systems

This proposal for a development methodology is geared toward production systems which are rule-based expert systems.

Steps in the Development Process

1) problem selection
2) requirements analysis
3) requirements specification
4) expert review
5) implementation
6) validation and verification

6.1 Problem Selection

Problem selection concerns the practicality of an expert system for a particular problem. Some example characteristics of problems suitable and unsuitable for expert system utilization were presented in section two of this paper. Reiterating, in order to extract their expertise, human domain experts must exist and they must be able to articulate their methods. If multiple experts are consulted for the system, they must agree on the solution process so that conflicting rules are not developed. The task must require only cognitive skills, must not be poorly understood, and must not be too difficult. If the experts do not thoroughly understand the solution process or it takes them weeks to solve it, then most likely an expert system will not capable of the

problem.

6.2 Requirements Analysis

This stage of development centers on defining the problem domain, acquiring the expertise to solve the problem, and specifying the appropriate system end results such as user interfaces, performance bounds, and validation criteria [5]. Through several interviews and/or observations of the expert, the knowledge engineer begins to formulate a high level logical view of the expert system preferably using the diagrammatical approach presented in the previous section. Upon preliminary designs, the engineer should review the plan with the expert for feedback of misconceptions and/or places for improvement. The expert should take an active part in creating the diagram so that clear, concise, and accurate details of the problem and the solution may be worked out.

Using the previous example of the card game hearts, diagram 2 was created as the initial activity diagram. However, suppose diagram 6 was created instead. This design is clearly inferior to the previous design. Several important steps have been left out of the design. The expert and knowledge engineer should recognize the fallacy of the design and reconstruct the diagram. It is essential to clearly define the process in such a way that both the expert and the knowledge engineer can visualize the system using the diagrammatical approach before attempting further development.

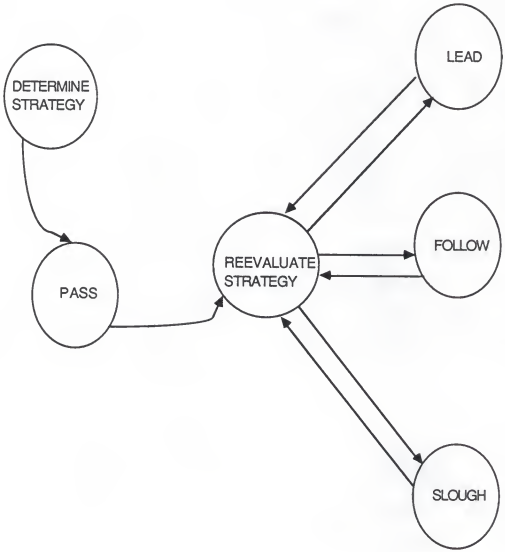

DETERMINE STRATEGY
PASS
REEVALUATE STRATEGY
LEAD
FOLLOW
SLOUGH

DIAGRAM 6

6.3 Requirements Specification

Once the requirements analysis phase has been completed, the requirements of the project should be specified in order to completely and formally describe the requested system. This proposal should include the complete set of diagrams, the knowledge representation structure to be utilized, the set of facts relevant, functional descriptions, interface specifications, and validation criteria.

6.4 Expert Review

Again to stress the importance of clarity and accuracy of the approach to the problem, the expert involved in the interview phase and other experts if available should review the requirements specification before implementation begins. Similar to peer reviews for large scale conventional projects, these reviews should address concerns such as achievability of proposed solution, ease of future maintainability, alternative approaches, and technical accuracy.

6.5 Implementation

The implementation phase of development centers on selecting the appropriate language for the problem and actually coding a working solution to the problem.

6.6 Validation and Verification

Verification of the implemented expert system ensures that the system functions as specified in the requirements phase. Given test cases, it performs within an

acceptable level as the human expert would. This of course encourages use of the expert again to verify that given the same set of data, both the human and the expert system come to the same conclusion.

Validation of the expert system ensures that it meets the needs of the user for whom it was designed. Will the expert system handle the typical cases occurring in the user's environment? Will it degrade gracefully as it reaches its limits of inference? Will it require extensive training to operate or is it self-explanatory through the user interfaces? Questions like these must be answered before the system is installed for the user.

6.7 Conclusions

As a final comment, the management of this process is a highly complex activity. As argued by Cupello and Mighelevich [15], if expert systems do not presently exist in the company, that technology should be acquired by the following steps:

1) the development managers should be technically trained with the appropriate AI background and have proven managerial skills.

2) the managers should report directly to an executive capable of funding and direction decisions for the company.

3) accumulation of personnel should be approached by training the best computer scientists in the company in AI techniques.

CHAPTER SEVEN

Conclusions and Future Work

This paper has presented a methodology of development for expert systems based on a logical diagrammatical design. Similar design diagrams have proven their advantages for preliminary designs in other areas. For example, conventional programming utilizes dataflow and hierarchical diagrams in development. Database designers utilize entity/relationship diagrams. Rather than using technical specifications and equipment, the graphical approach is more readily understandable for non-technical personnel. Thus, both the designer and the customer can communicate on common ground.

Most of the current literature agrees that a major problem exists in systems characterized as event driven [16]. "Examples include telephones, communication networks, computer operating systems, avionics ... " and expert systems. Attempting to solve this problem, graphical approach methods such as petri nets, sequence diagrams, temporal logic, and statecharts, "a higraph-based extension of standard state-transition diagrams," have been used. Statecharts were developed to overcome the non-refinement aspects of state diagrams, to allow a notation for concurrency, and to restructure the drawing of the same transition occurring in several states.

However, stepwise refinement of the expert system development is possible using the activity and rule diagrams presented. The activity and rule diagrams more clearly

enhance expert system development through a step-wise refinement of the system into the activities and the rules decomposed from the activities. Harel's method can be used as a means of clarifying some system diagrams.

If the field of expert systems is to advance, an improved development methodology must be employed. The diagrammatical approach presented in this paper is a valid option. A framework of development can be built based on this approach. Once we have established expert systems as a science, we can further study and improve the techniques involved. The diagrammatical approach is the first step.

Future research considerations may focus on the applicability of other software engineering techniques to the diagrams and expert system development. An interesting idea is the study of applying McCabe's complexity measures to the diagrams as a basis for judging complexity and reliability of the system. Another idea is the possibility of constructing test sets from the diagrams.

Empirical tests will be needed to further study expert systems development and its role in computer science. Expert systems show promise as a viable alternative paradigm to be considered. But, we must standardize the development process so that correct, efficient, and reliable systems can be produced in a cost effective manner.

BIBLIOGRAPHY

[1] Forsyth, Richard. Expert Systems, Principles and Case Studies. Chapman and Hall Computing, New York. 1984.

[2] Waterman, Donald A. A Guide to Expert Systems. Addison-Wesley Publishing Company, Massachusetts, 1986.

[3] Brownston, Lee, Robert Farrell, Elaine Kant, and Nancy Martin. Programming Expert Systems in OPS5. Addison-Wesley Publishing Company, Massachusetts, 1985.

[4] Hayes-Roth, Frederick. "Rule-Based Systems." Communications of ACM, September 1985, volume 28, number 9, pp 921-932.

[5] Pressman, Roger S. Software Engineering A Practitioner's Approach. McGraw-Hill Book Company, New York, 1982.

[6] Wright, Peggy A. "A Proposed Curriculum for Software Engineering of Knowledge Based Systems." 2nd Annual Conference of Knowledge Based Systems, 1984.

[7] Allen, Elizabeth. "YAPS: Yet Another Production System." Maryland Artificial Intelligence Group, December 1983.

[8] Bobrow, Daniel G., Sanjay Mittal, and Mark J. Stefik. "Expert Systems Perils and Promise." Communications of ACM, September 1986, volume 29, number 9, pp 880-894.

[9] Geissman, James R. and Roger D. Schultz. "Verification and Validation of Expert Systems." AI Expert, February 1988, volume 3, number 2, pp 26-33.

[10] Duda, Richard O., Peter E. Hart, Rene Reboh, John Reiter, and Tore Risch. "Syntel: Using a Functional Language for Financial Risk Assessment." IEEE Expert, Fall 1987, pp 18-30.

[11] Zaulkerman, I., W.T. Tsai, and D. Volovik. "Expert Systems and Software Engineering: Ready for Marriage?" IEEE Expert, Winter 1986, pp 24-30.

[12] Leinweber, David. "Expert Systems in Space." IEEE Expert, Spring 1987, pp 26-36.

[13] Alvey, P.L., C.D. Myers, and M.F. Greaves. "An Analysis of the Problems of Augmenting a Small Expert System." Proceedings of the Fourth Technical Conference of the British Computer Society Specialist Group on Expert Systems. University of Warwick, December 1984, 61-72.

[14] Hoyle, Edmond. The Official Rules of Card Games. The United States Playing Card Company, 1961, pp 129-132.

[15] Cupello, James M. and David J. Mishelevich. "Managing Prototype Knowledge/Expert System Projects." Communications of ACM, May 1988, volume 31, number 5, 534-541.

[16] Harel, David. "On Visual Formalisms." Communications of the ACM, May 1988, Volume 31, number 5, 514-530.

Structuring the Development of Production Systems

by

Rodney D. Anderson

Kansas State University, Fall 1988

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1988

ABSTRACT

This thesis presents a diagrammatical approach to the development of rule-based expert systems. Expert systems have proven themselves useful in commercial environments, but, the development process needs to be improved. The method presented in this thesis utilizes an activity diagram to represent the various activities and transitions among the activities in the system. Decomposition of the activities can be performed to produce the rules and transitions among the rules. This diagrammatical approach produces a structured knowledge base that is easier to understand, maintain, and enhance than an unstructured knowledge base. This approach also supports a development methodology that may lead to the same advantages such as those realized in conventional programming using software engineering techniques.