Automated Knowledge Engineering

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Maanda Raudzingana

Grahamstown, South Africa
October 2014

Abstract

The knowledge acquisition process is a critical one in the development of knowledge based systems. As the name suggests, these systems rely heavily on complete and accurate knowledge. Consequently, the credibility of the knowledge stored in the knowledge base has a direct influence on the performance of the overall system. For this reason, it is primary that the process is conducted with caution and diligence. However, this is often not the case owing to the difficulties inherent in the traditional approach of knowledge acquisition, which involves physical collaboration between the domain experts and the knowledge engineer. Automating the process would enable the efficient creation of practical knowledge-based systems.

The development of a modularized tool to enable the automated construction of knowledge bases is discussed. This allows the construction and extension of knowledge bases by domain experts, thereby eliminating the overhead common in the knowledge acquisition phase. The results obtained during evaluation confirm the credibility of the generated output and hence, the applicability of the developed tool.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (2012 version, valid through 2013):
H.3.4 [Decision support systems]: Expert systems
L.4.5 [Health care information systems]
General-Terms: Human Factors, Verification

Acknowledgements

I would like to extend thanks to my supervisor, Dr Karen Bradshaw for her support and guidance through out the duration of this research project. Her guidance and support was key to the success of this project.

I would also like to thank the Computer Science department at Rhodes University for providing the facilities used through out the year, thereby making the completion of this project a reality.

A special thanks to friends and family for the support given through out the year.

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Tellabs, Genband, Easttel, Bright Ideas 39, THRIP and NRF SA (TP13070820716). The authors acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author(s) and that none of the above mentioned sponsors accept liability whatsoever in this regard.

Contents

1 Introduction **9**

1.1 Problem statement 9

1.2 Research goal 9

1.3 Document structure 10

2 Literature Review **11**

2.1 Expert Systems 11

2.1.1 Applications of Expert Systems 12

2.1.2 Components of an Expert System 12

2.2 Knowledge Representation 17

2.2.1 Knowledge Acquisition 18

2.2.2 Knowledge Representation Schemes 19

2.3 Examples of Expert Systems 25

2.4 Attempts at Knowledge Base Automation 29

2.5 Tools and Techniques 35

2.5.1 Client Interface 36

2.5.2 Intermediate Representation 39

2.5.3 Translation 40
2.5.4 Testing 42
2.6 Summary 43

3 Knowledge Elicitation 44

3.1 Overview 44
3.2 Knowledge Elicitation 46
3.2.1 Information source 47
3.2.2 Data elements of interest 47
3.2.3 Storing of acquired data 48
3.2.4 The user interface 49
3.3 Intermediate Knowledge Generation 53
3.3.1 XML Generetor 53
3.4 Summary 58

4 Rule Generation 59

4.0.1 Rule-based representation scheme 59
4.0.2 Problem domain 60
4.0.3 Syntax of rule base 60
4.0.4 Automated generation 63
4.1 Summary 72

5 Evaluation of the Knowledge Base 73

5.1 Success criteria 73

5.1.1 Representing knowledge as rules 74

5.1.2 Reaching valid conclusions 74

5.1.3 Execution by suitable inference engine 75

5.2 Manual evaluation 75

5.3 Evaluation using VP-Expert shell 79

5.3.1 Running the VP-Expert shell 80

5.3.2 Test cases 80

5.3.3 Limitations of VP-Expert expert system shell 86

5.4 Discussion 87

5.5 Summary 88

6 Conclusion 89

6.1 Research objectives 89

6.2 Future work 90

List of Figures

2.1 Components of an expert system 13

3.1 Steps, input, and outputs of the modules involved in the knowledge base generation process 45

3.2 Screenshot of the KAT GUI 50

3.3 Screenshot of dialogue confirming successful disease entry 52

3.4 Screenshot of error dialogue alerting the user of missing data 53

5.1 Structure of proposed inference engine 76

5.2 Screenshot of VP-Expert interface showing an Anthrax diagnosis 83

5.3 Screenshot of VP-Expert interface showing an Botulism diagnosis 84

5.4 Screenshot of VP-Expert interface showing an Undecided conclusion 86

List of Tables

5.1 Results of inference step by the inference engine 79

5.2 Distribution of symptoms across diseases. 82

Listings

3.1 Human-readable representation 48

3.2 Invoking the XML-generator 55

3.3 Generation of XML-based representation 56

3.4 Extraction of a disease name 56

3.5 Extraction of primary symptoms from a disease 57

3.6 Sample output of the XML generator 57

4.1 Arrangment of different blocks 62

4.2 Example of a simple knowledge base file 62

4.3 Rule type 1 63

4.4 Rule type 2 64

4.5 Rule type 3 64

4.6 Rule type 4 64

4.7 Invoking of the rule generator 65

4.8 Method definition for Generate method 66

4.9 Method definition for ShowActionsBlock method 67

4.10 Construction of a rule of Rule Type 1 69

4.11 Method definition for showQueryBlock method 70

5.1 Example rule in the generated knowledge base 74

5.2 Example rule for executation by hypothetical inference engine 77

5.3 State of working memory before and after inference 78

5.4 Querying of symptoms by VP-Expert 82

5.5 Querying of symptoms by the expert system in the second test case 84

5.6 Querying of symptoms by the expert system in the third test case 85

Chapter 1

Introduction

1.1 Problem statement

Knowledge engineering or knowledge acquisition is a critical process in the development of expert systems. It is the process by which expert knowledge is integrated into a knowledge-based system. For this reason, a substantial amount of research has been conducted in an attempt to come up with more efficient and effective ways to go about this process. However the process of knowledge engineering has inherent difficulties. These include time constraints on the expert's side hence rendering the expert inaccessible. In addition, experts are unenthusiastic and usually, there exists a lack of communication between the knowledge engineer and the expert. Furthermore, it has proved time consuming and tedious to have a knowledge engineer reconfigure the knowledge base whenever new facts and knowledge became available.

1.2 Research goal

The research aims to investigate the use of knowledge base construction tools available, their strengths and weakness as well as current trends and issues in knowledge representation. Special emphasis is placed on the automation of knowledge base construction or knowledge representation using the production system approach. This raises the question: can the knowledge representation phase in expert systems development be automated with the help of modern technologies? The research aims to find out if it is possible to eliminate entirely the need for a knowledge engineer by automating the knowledge engineering

process. This requires the automation of processes involved, i.e. vocabulary construction, classification of domain objects and attributes attributes thereof, discovery of parameter relationships and inter-dependencies, representation of core facts within the domain and finally the presentation of a rule-based knowledge representation that can be executed by a suitable inference engine [9]. With that said, the primary aim of the project is to present an efficient approach to knowledge representation in the development of expert systems by automating the knowledge engineering process. This will be achieved by developing a modularized tool to enable the automated construction of knowledge bases. The generated knowledge base should *1)* successfully represent as rules the knowledge gathered from domain expert knowledge sources; *2)* allow execution by a suitable system, and *3)* allow the executing program to reach valid conclusions. Over and above, the developed tool should allow the extension of the generated knowledge base by domain experts.

1.3 Document structure

The research paper firstly discusses the findings from background research. This discussion focuses on the concept of an expert system. Next, different representation schemes are discussed. Several knowledge acquisition tools are discussed next, and finally, the different tools to aid the development of acquisition tools are discussed. This makes up Chapter 2 of the thesis.

The design and implementation of the preliminary modules of the developed system is discussed in Chapter 3, focusing on the input, operation, and output of the modules.

Following the discussion on the preliminary modules, the design and implementation of the core module of the system is discussed. The generation of rules is discussed, focusing on the input of the rule generator, the different rule types implemented, and the structure of resulting knowledge base. This makes up Chapter 4 of the thesis.

Chapter 5 discusses the evaluation of the output of rule generator, the actual knowledge base. Results obtained from test cases are discussed.

Lastly, the outcome of the projects is given and coordinated with the project objectives. Future work is suggested in an attempt to improve applicability of the research project with regard to knowledge engineering.

Chapter 2

Literature Review

The aim of this chapter is to review issues around the development of expert systems. With emphasis given to the knowledge representation process. An overview of expert systems highlights the foundation and necessity of improvements in the representation of knowledge. It has been argued that failure to represent effectively appropriate amounts of knowledge in knowledge based systems results in limited performance during consultation [36]. The discussion highlights the implications, and research and development trends in knowledge acquisition. Knowledge representation schemes and their characteristics are presented. Research efforts have influenced the development and refinement of automated knowledge engineering tools. A few of these are reviewed. Finally, the tools and techniques available for the development of these knowledge engineering tools are explored.

2.1 Expert Systems

An expert system [24] is an example of a knowledge based system whose primary purpose is to mimic human reasoning. This is achieved by emulating the decision-making ability of a human expert in a specific domain. A structured organization of knowledge, facts and reasoning techniques is used to solve problems that would normally require human expertise. The organized collection of facts and knowledge is known as the knowledge base. This essentially houses the knowledge transferable to a user during consultation. The expert system applies the facts stored in the knowledge base to arrive at a conclusion. This is the purpose of the inference engine. Facts together with information provided by the user are interpreted and evaluated to infer new knowledge. A representation language

is used to express facts [19], with common representations including predicate calculus, rules, graphs and decision trees . Expert systems are considered to be successful because they are successful at transferring expert knowledge in a given problem domain. For this reason, expert systems have seen widespread use in several domains. Expert system technology is currently used to solve many industrial and commercial problems.

2.1.1 Applications of Expert Systems

Applications of the technology can be clustered into classes [20] such as diagnosis, troubleshooting, prediction, configuration, decision making, knowledge publishing, monitoring and control, design and manufacturing as well as debugging and instruction. Systems in these classes often employ highly structured rules to enable them to achieve the decision-making ability similar to that of a human. Successful examples include medical diagnostic systems that are in use today in several medical facilities, computer network diagnostic systems used by network administrators for network fault diagnosis, mission control systems used by military personnel, and the training and crisis management systems used by different organizations. Recent literature has highlighted the apparent interest in integrating expert systems technology with existing applications in various domains [19]. These include multimedia applications, accounting and business management applications, and applications in the medical and educational domains. This widespread use of expert systems technology highlights the relevance of expert systems research and development in this day and age.

2.1.2 Components of an Expert System

Three basic components make up an expert system: a knowledge base, an inference engine, and a user interface. Some systems have a knowledge acquisition module to allow domain experts to add new knowledge into the knowledge base. Shown in Figure 2.1 are the different components of an expert system.

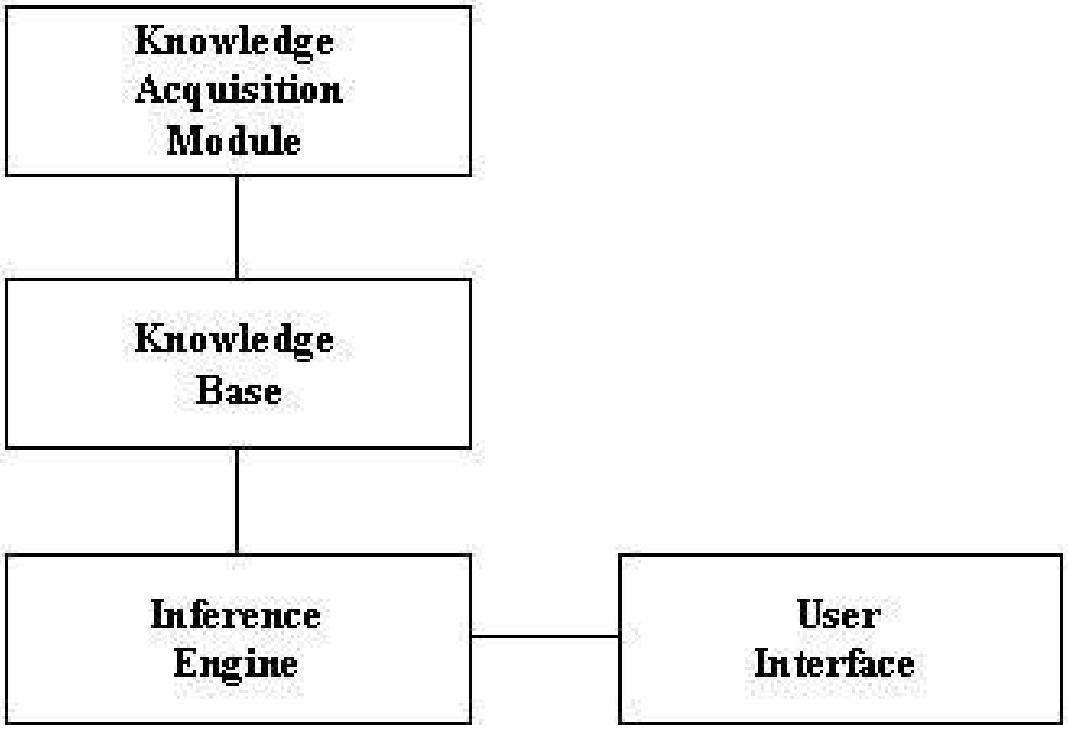

Figure 2.1: Components of an expert system

These components work together to effectively transfer declarative and procedural knowledge to the user upon consultation [42].

The Knowledge Base

The knowledge base is the store of facts and rules. It contains the domain knowledge vital for problem solving, i.e., declarative knowledge. Knowledge is organized as facts, objects, attributes and conditions [24]. During inference, the knowledge base is searched and manipulated according to predefined rules of logic. The knowledge contained in the knowledge base is commonly acquired through transfer from a human expert using appropriate methods. The system thus represents the expert's knowledge and expertise in the specific domain. A human expert is known to possess extensive knowledge on a given class of problems. Embedding such limited knowledge in a computer system has proven to be an easier task compared to embedding large amounts of information from different classes of problems [13].

Expert systems transfer human expertise. This has been described as intensive knowledge specific to a domain [19]. According to Iancu et al. [19], the knowledge can be categorized

into knowledge types: facts and theories about the domain, rules and procedures, global strategies on solving problem types and finally, meta-knowledge.
Part of the contents of the knowledge base is facts. These are known items of information of a given subject. In a medical scenario, the statement: "Streptococcus pneumoniae causes Pneumonia" can be considered a fact. This is information that is known and can be proven. A collection of facts is housed in the knowledge base, which describe a given subject in detail.
Besides facts, contained in the knowledge base are objects [24]. Objects are representations of real world entities, perceptible by one or more senses. In a medical context, Pneumonia can be classified as an object. These objects have attributes that describe them. The use of the facts and objects by the inference engine is governed by a set of conditions or rules.

Uncertainty

It is rare that knowledge is certain. The incomplete, ambiguous and uncertain nature of information introduces difficulties during the development of intelligent systems [36]. This introduces the need for expert systems to handle uncertain knowledge. In the early days of expert systems development, marking a milestone in the development thereof, was the introduction of a level of certainty in accumulation of evidence and confidence of the hypothesis. These came to be known as certainty factors [8]. Certainty factors allow an expert system to express a relative level of confidence, with a high factor denoting an absolute level of confidence and a low factor denoting the least level of confidence. A common approach to estimation of certainty is the use of probability theory. Sub-options include objective probability, which makes use of frequency refraining from any interpretation or degrees of confidence, and subjective probability, in which the lack of certainty is quantified, hence reflecting a degree of belief. Common practice is to use either approach in conjunction with other estimation mechanisms customized to a suitable extent. The PROSPECTOR system [13] handles uncertainty by assigning probabilities to conclusions using established statistical rules and theories. However, the use of statistical rules and theories introduces additional complexities with regard to statistical independence and prior probabilities. In an attempt to avoid such complexities, a novel calculus of certainty values was used to handle uncertainty in the MYCIN system. A statistical approach remains the most widely adopted approach to handling uncertainty during expert systems development. Despite successes of most implementations in handling uncertain knowledge, a point is yet to be reached where reasoning in the presence of vagueness and

ignorance as well as recognition of the limits of knowledge bases by systems is possible [13].

The Inference Engine

The inference engine [24] is the heart of an expert system. Invoked whenever a user performs a query, the inference engine's main purpose is to draw inferences from the supplied information and the information contained in the knowledge base. Logical rules are applied to the knowledge base in order to infer new knowledge [8]. This involves effective comparison of available information, searching of goals and causal relationships and finally the evaluation of the relative certainty of facts [24]. The inference engine makes use of forward chaining and backward chaining strategies to go about inference.

Forward chaining

Forward chaining [8; 24] is one of the techniques used by the inference engine during inference. This method starts from known data and proceeds with the data effectively using inference rules to extract more data until the goal is reached. For this reason, it is also known as data-driven reasoning. The engine searches the rule set for a rule whose condition is satisfied and then infers the action effectively adding a new fact to the data store. This method is favored for its ability to trigger new inferences as a result of the reception of new data [16]. However, as a result of using forward chaining, more rules than necessary may be executed resulting in an inefficient inference process.

Backward chaining

Backward chaining [8; 24], often used as an alternative or in conjunction with forward chaining, is another technique for inference. Backward chaining starts from the goal and attempts to find the evidence to prove it. This is also referred to as goal-driven reasoning. The rule set is searched to find rules whose action parts or consequents match the goal. When found, the rule is fired and the goal is proved. This process involves adding rules to a list of goals that will have to be proven in order to prove the initial goal. The engine sets aside the rule it is currently working with and then establishes a new goal to prove the condition part of the rule. Searching takes place for rules that can prove the sub goal. The process continues until no rules can be found that prove the current sub goal.

The choice of method between forward and backward chaining is arguably dependent on the problem domain in which the resulting system will be used.

The User Interface

To achieve effective transfer of knowledge from the system to a user during consultation, an interface needs to be present. The user interface is the component that allows for communication between the user and the system [24]. This makes use of a natural language interface to simulate causal conversation [38]. It enables the input of facts about a given situation that the engine can use together with information stored in the knowledge base to arrive at conclusions. Furthermore, the interface allows the user to query the system for any information the user needs, like why a specific question has been asked. It is crucial that the interface is simple and easy to use. This stems from the assumption that expert systems are used by non-technical users who may not have an in-depth understanding of the inner workings of the system. It is for this reason that the interface should resemble a human as far as possible to guarantee effectiveness and ease of communication. Duda et al. [13] refers to this as a characteristic of effective consultants, that is, the ability to maintain a model of the user and effectively assess what the user does, does not know, and what he or she is trying to achieve.

Knowledge Acquisition

The knowledge contained in the knowledge base is acquired from human experts. The process of acquiring knowledge and transfer thereof into the knowledge base is commonly referred to as the knowledge acquisition process [12]. Most expert systems have a knowledge acquisition component that facilitates the transfer of knowledge from human experts to the expert system. This is achieved through the presentation of an interface that makes possible a dialogue between the system and the expert [24]. The knowledge acquisition interface is sometimes the user interface a non-expert user would use to query and supply information to the system during consultation, however, specific to the task of knowledge acquisition.

Alternative ways of knowledge acquisition include interviewing domain experts. The knowledge is collected and then translated into appropriate symbolic representations by a knowledge engineer. This was the common approach in the early days in expert systems development. However, research and development of new technologies has introduced

tools and techniques that aid the automation of the knowledge acquisition process. Such tools are discussed in subsequent sections.

Explanation Facility

Besides the three basic elements discussed above, more sophisticated implementations include additional elements to improve the intended functionality of an expert system. One of these elements is the explanation facility, which most expert systems have [24; 8]. This enables the user to query the system on certain information. Examples include explanations on why certain information is needed, and how a particular conclusion was reached. An expert system should then be able to explain the reasoning behind its analysis and conclusion. This further enables the user to monitor the system as it processes rules [24]. Special commands are defined that cause the system to invoke this explanation functionality. In most expert systems, during consultation as the user supplies information to the expert system, it is possible to request an explanation on why a given question is being asked. The expected response to this is a display of where and how the piece of information is used. In most cases, this is an argument required for a given rule to be fired. An explanation of how a conclusion was reached is often required. The response takes the form of a chain of rules the system has applied to arrive at the reached conclusion. The necessity and importance of the ability of a system to explain its processes justifies the need for multiple levels of representation [8]. Such behavior further highlights the success of expert systems in mimicking human behavior in decision making; human experts are often required to provide explanations of their reasoning.

2.2 Knowledge Representation

Knowledge representation [8], considered by many be to the key issue at this point in the development of AI [33] is the area of AI that involves the transformation and translation of facts and information about a given subject into a structured symbolic form that allows manipulation in an automated way by reasoning programs. These facts are stored as knowledge in the system's knowledge base. The representation process is concerned with the encoding of the acquired knowledge into the knowledge base. For this reason it is a critical process in expert systems development. Research has focused on trying to find ways to represent knowledge that require less effort but maximize the system's performance. The representation of knowledge in notable systems was reported to be a

major contributing factor to a large proportion of reported errors [13]. An effective representation has to address certain issues. Duda et al. [13] demonstrates that a knowledge representation formalism should effectively represent the concepts and intentions of the expert, and allow correct interpretation by reasoning programs. Furthermore, it should support explanations that reflect human reasoning and be capable of identifying gaps in the knowledge as well as allow the separation of domain knowledge from the interpretation program to allow independent alteration of either. Such demands introduce difficulties in the process of knowledge representation, which if addressed would make way for improvements in the development of expert systems. Among these are issues on knowledge acquisition and the ensuing representation of the acquired knowledge.

2.2.1 Knowledge Acquisition

Knowledge acquisition is concerned with the transfer of knowledge from the expert into an intermediate store that can subsequently be translated into a formalized representation. Acquisition usually takes places in one of two ways: manually through interactions between an expert and a knowledge engineer, or automatically through the use of appropriate automated systems. Several methods are used to acquire knowledge from an expert. These include interviews, questionnaires, surveys and web-based systems that present an interface allowing experts to input knowledge into the system. However, the most common method of acquisition involves communication between a knowledge engineer and human experts. The knowledge engineer's responsibility is to program the knowledge into the expert system. This requires use of different knowledge representation schemes that present an executable representation to the inference engine or interpreter for execution.

Knowledge acquisition in its own right is time consuming and consequently poses costs during development of expert systems. Automating the knowledge acquisition process is a commonly proposed solution [15]. Automated knowledge acquisition tools have seen widespread use in recent years, with most of these under continuous refinement. These are discussed in a later section. Automated acquisition tools encourage interaction between experts and the system. Use of these requires that the experts receive training on how to use the system. Automated knowledge acquisition tools enable the knowledge engineer and domain experts to build and maintain the resulting knowledge systems [36]. Notable features used in the classification of such tools include the degree of automation, the dependency of the domain, problem type and user type.

Problems with knowledge acquisition have contributed significantly to the deterred improvement of expert systems over the years. Duda et al. [13] notes the incompleteness of the knowledge bases of the MYCIN and PROSPECTOR systems as stemming from problems in knowledge acquisition. Often the construction of knowledge bases, involving the identification and encoding of knowledge, reveals gaps and weakness in both the understanding of the problem domain and the representation techniques [13]. Rafea et al. [36] emphasizes the importance of a well functioning knowledge acquisition tool in expert systems development endeavors. Often, consultation performance is limited as a result of failure to acquire and encode relevant knowledge, which warrants diligence during the construction of these tools. Ford et al. [15] maintain that the knowledge acquisition process extends for the life cycle of the system. It is stated that at any given instance, an acquisition tool should support directly both the domain expert and the knowledge engineer. With this, Ford et al. [15] note the different facets of the knowledge acquisition process, which encompasses the elicitation and modeling of human expertise, testing of the systems efficacy, refinement of the knowledge base, maintenance of the resulting system, and elaboration of an explanation capability. These facets highlight the importance of the knowledge acquisition process in the development of consultants.

2.2.2 Knowledge Representation Schemes

During development, a knowledge engineer must choose the format to be used in representing the knowledge. A number of different formats are available with different structures and characteristics and as result fare differently in representing different kinds of knowledge.

Semantic Networks

A semantic network [24] is a model of associative memory, a graphical knowledge representation method that uses patterns of interconnected nodes and arcs. The nodes effectively represent objects, with the arcs representing the relationships between the objects. The idea behind semantic networks is that knowledge can be stored in the form of a graph. Semantic networks are widely used as a knowledge representation technique owing to their robustness and guarantee of efficiency in searching and manipulation of the data structure. The relationship with other connecting objects or nodes gives meaning to an object, and hence allows for activation of an object on a given node. A simple semantic network in the LISP [26] language could be defined as:

```
(defun *database* ()(
(human  (has-a house) (color white) (size big))
   (house   (has-a kitchen))
    (kitchen    (is-a room)(has-part stove))
    )
 )
```

In the example above, humans, house and kitchen are objects. The arcs are represented by the association functions: have-a, is-a, and has-part. The attributes of the *house* object in this case are color and size. New nodes and arcs can be added to a network at any time during the development process. These data structures have been widely used in a variety of ways. Tanwar et al. [41] demonstrates that there are six common kinds of networks used in various applications, namely, definitional networks, assertional networks, implicational networks, executable networks, learning networks and hybrid networks.

Definitional networks represent relations between a concept type and a newly defined subtype, like the relation: an apple is-a fruit. These are known to support the rule of inheritance. Assertional networks are used for the assertion of propositions, like, if a man owns a car, then he drives it. Implicational networks are commonly used to represent patterns of beliefs, causality, or inferences. They use implication as the relation to connect nodes. An example would be: if the grass is wet, then it is slippery. Executable networks present mechanisms to perform inferences and or search for patterns and associations. Mechanism include attached procedures, graph transformations, and message passing. Yet another example is learning networks. These use acquired knowledge to build their representations. Knowledge is acquired from examples. When new knowledge is introduced, extension or deletion of the old contained knowledge takes place. Hybrid networks are made up of a combination of any of the networks discussed above, either in a single network or in separate networks [40].

The presence of links between objects in semantic networks, synonymous to mental links humans form between real world objects, presents semantic networks as a representation that closely resembles human knowledge structuring. Meaning is derived from the presence of and connection to other objects. Semantic networks are favored for their ability to facilitate the inheritance of data that prevents duplication of data, as well as their ability to allow efficient manipulation. Argued to be the most useful form of inference,

inheritance allows elements of a class to inherit characteristics and behavior from a class higher in the hierarchy. Multiple inheritance introduces additional benefits, that is, objects can belong to more than one class, and in turn, a class can be a subset of more than one class. Additionally, semantic networks fare well in representing knowledge involving related terms, and in conveying meaning in a transparent manner. Their simple and easily understood structure makes it easy to translate them into a formalized symbolic representation that can be executed by programs.

Frames

A frame [8] is an AI data structure used for the representation of data. Two types of frames exist: individual frames, used for the representation of single objects, and generic frames, for the representation of classes of objects [8]. Each individual frame represents an object or situation and consists of a group of attributes that describe an object or situation. A frame consists of slots in which the attributes are stored. These attributes contain values and procedures that act on the values. Illustrated below is the schematic representation of a frame.

```
(Frame-name
<slot-name1 attribute1>
<slot-name2 attirbute2>
..)
```

The attribute values can be numbers, strings, or identifiers of other frames. Frames can be linked together, which is called inheritance. Upon retrieving a frame, an agent changes some of the information in the frame. This information can be declarative or procedural. A declarative representation, representing objects, facts and relations asserts the validity of an attribute whereas a procedural representation, representing the actions performed by objects, contains instructions to be executed in a given slot. With this, the frame provides information storage and instruction on how to act in a given situation. Information types stored in frames include: relationships between the frame and other frames, information for choosing frames, blank slots and procedures to be executed after various slots have been filled. Frames are favored for their structure, which allows them to be programmed and manipulated using object-oriented programming tools.

Frames, like semantic networks have the advantage of allowing reuse of related information through inheritance, thus eliminating the need to develop repetitive blocks of knowledge data. Frames fare well in representing knowledge that can be stored in chunks [24].

First-order logic

First-order logic [24], also known as first-order predicate calculus, propositional calculus, and predicate logic, is a method used for capturing declarative knowledge.

In first-order logic, a sentence is made up of two parts, a subject and a predicate. A predicate, which defines the properties of the subject can only refer to a single subject. Sentences take the form P(x), where P is the predicate and x is the subject, represented as a variable [37]. Manipulation takes places on logically combined complete sentences according to the same rules as those used in Boolean algebra. Universal (e.g. for all) and existential (e.g. for some) quantifiers are used to structure sentences. For example, given a variable x and predicates A and B, first-order logic allows the construction of sentences like:

$\forall x$: Ax Bx

The above translates to: "For all x, if x is A , then x is B. The level of ease with which sentences can be constructed using such syntax has resulted in AI researchers taking interest in first-order logic as a means of representing knowledge. Subsequently, computer programs have been developed using first-order logic [37].

First-order logic is favored for its declarative nature which allows facts to be represented within the syntax. Furthermore, it allows operations on information unlike most data structures. Examples of operations include dis-junction and negation. It further allows for the assignment of context-independent meaning, unlike natural languages where meaning is context-dependent. However, first order logic falls short in that it has limited expressive power [5]. A famous argument in logic can be used to illustrate this.

```
All A are B.
All B are C.
Therefore, all A are C.
```

It is argued that no good way exists to express the above using propositional calculus [34].

Rules

Rules or production rules [24] are perhaps the most prominent method for representing knowledge. Systems that employ rules for representation are commonly referred to as rule-based systems or production systems. Production systems maintain an ongoing store of assertions in working memory [8]. A production rule is made up of condition-action or antecedent-consequent pairs. A given pair represents an IF-THEN structure that links supplied information in the IF part to an action in the THEN part. The simple structure and syntax of rules makes them human readable and easy to implement. A rule in its own right provides some description of how to solve a given problem. It is possible to extend the antecedent part of a rule by adding more of these. This is achieved by adding keywords like AND for conjunction and OR for disjunction, or a combinations of these. An antecedent contains an object linked to a value by an operator. Mathematical operations are often used with rules on objects and values that permit mathematical manipulations. Examples include comparisons, additions, logical operations and boolean operations. Rules can be used to represent relations, recommendations, directives, strategies and heuristics. Depending on the amount of knowledge to be represented, rule structures range from simple antecedent-consequent pairs to a large complex set of complicated rules. An example of a rule is given below.

```
if<antecedent> then <consequent>

IF day_of_week = 'Sunday'
THEN lectures = False
```

The given example shows how a rule can be used to represent relations. According to this specification, whenever the day of the week is Sunday, lectures should be set to false, which would simply imply no lecture attendance, as is the case in most academic institutions. In this case, *day_of_week* is a variable whose value would have been supplied by a user. The system uses the value to determine whether there should be lecture attendance. During interpretation, the rule set is searched for a rule whose condition is satisfied. When found, the action is invoked. Although simple, this example highlights the basic procedure of rule interpretation by a rule-based system. More complex rules are common in large systems. For example, the MYCIN system had a rule-based knowledge base consisting of five hundred rules [39]. This can be argued to be a reasonable amount considering

MYCIN's performance during consultation.

Uncertainties can be expressed in rules by attaching certainty factors to the antecedent, consequent, or both [24]. Implementations often make use of custom methods to handle uncertain knowledge. These methods mostly include assigning a level of confidence to a given rule. This requires customization of the knowledge base to allow expression of the confidence of the expert system.

Hybrids

A common approach to knowledge representation is the use of more than one knowledge representation scheme. The systems are customized to leverage the functionality and benefits of two or more representations formats. This has been known to introduce improvements in knowledge representation and in most cases, inference. Tanwar et al. [41] highlights the strength of a hybrid knowledge representation technique. A technique is described, which uses semantic networks and scripts to represent knowledge. In this experiment, scripts complement the representation of non-event based knowledge by filling the gaps resulting from incomplete or disjoint observations. Despite improvements introduced by hybrid representation techniques, the use of two representation techniques introduces complexities. For this reason, it is encouraged to adopt a hybrid approach with extreme caution and complete understanding of the intended functionality.

Comparison of schemes

Niwa et al. [33] presents a comparison of four knowledge representation schemes: simple production systems, structured production systems, frame systems and logic systems, based on the difficulty of the implementation and runtime efficiency. The applications domain used was the risk management of large construction projects. Implementations of the representations would be tested on four systems developed using a modified version of the LISP programming language. Common components of all the systems were an inference engine, and a knowledge maintenance function to facilitate alteration of the knowledge base. The systems were capable of both forward and backward chaining reasoning. Forward chaining was used to inform the user of consequent risks that could follow

specific causes, whereas upon input of a risk hypothesis by the user backward chaining was used to query the user about conditions until the hypothesis, in this case the risk, was proven. After testing, findings revealed that the level of difficulty of implementing a knowledge base was directly proportional to the degree of structuredness. However, runtime efficiency was found to increase with increased structuredness. Structuredness was also found to decrease sensitivity to the size of the knowledge base [33]. Findings further revealed that inference was slow with logic systems used as representations. This lead to the conclusion that logic systems, compared to the other representations are less efficient and difficult to implement owing to mathematical completeness. Use of modular knowledge, that is, simple production systems and logic systems, is encouraged in poorly understood domains with poorly structured knowledge, although this has the cost of low runtime efficiency. Structured knowledge representations guarantee maximum runtime efficiency at the cost of difficulties during implementation.

Summary

The process of knowledge representation has proven to be the bottleneck of expert systems development[12; 25]. Iancu et al. [19] argues that the problem in knowledge problem is the introduction of information in systems that permits easy access and usability in solving problems. Duda et al [13] predicts significant advances in the capabilities of expert systems when the unsolved problems in areas such as knowledge representation and learning are solved. Efforts have been directed toward establishing alternative ways of acquiring and representing knowledge using the least resources yet still guaranteeing effectiveness [12]. This has lead to the emergence of knowledge acquisition and representation tools. Achieved functionality includes automation to varying degrees, of the sub-processes involved.

2.3 Examples of Expert Systems

As discussed, the main purpose of an expert system is to transfer human expertise within a specific domain to users who may not have such knowledge. This can be done in a number of ways. Notable examples are: rule based expert systems which use knowledge bases containing expert knowledge in the form of IF-THEN rules; case based reasoning systems, which aim to solve new problems by reusing already established solutions used to

solved previous problems; fuzzy expert systems, which aim to model uncertainty through the use of fuzzy membership functions and rules to reason about data; and neural network systems, which solve problems by simulating the biological structure of the human brain and nervous system [3]. Among the the first truly successful forms of AI software [43] and despite their limited knowledge and versatility, expert systems have achieved remarkable levels of performance in designated domains [13]. The expert systems domain has a wide area of application [19]. In an attempt to highlight the successes in various domains, a description of various expert systems, their domains, and characteristics are given below.

The MYCIN System

MYCIN [39] is an expert system developed at Stanford in 1972 for the purpose of treating blood infections. Reported to have operated at the same level of competence as specialists in blood infections, it was capable of identifying the bacteria causing infections, proceeding to recommend antibiotics and prescribing a dosage taking into account the patient's body weight. It used reported symptoms and medical results in an attempt to diagnose patients. MYCIN could request further information on a patient and was capable of suggesting additional tests in an attempt to arrive at a diagnosis. It could, at request, explain the reasoning behind a specific diagnosis. It gathered information from the user by asking a series of questions. At the end, it would give a list of possible answers, i.e., its guess on the bacteria causing the infection, together with the certainty factor associated with each diagnosis. MYCIN used a goal-oriented strategy to reason from an initial goal. With a fairly simple inference engine, the system's success was due to its effective use of certainty factors together with the implementation of a sophisticated knowledge representation and reasoning scheme, which influenced the development of subsequent rule-based systems following MYCIN's introduction of the approach [39]. MYCIN was never used in practice for ethical reasons, not including poor performance.

The R1 System

R1 [27], also known as XCON is a rule-based expert system for configuring VAX-11 computer systems. Developed at Carnegie Mellon and DEC in the late 70's, it has the ability to display diagrams showing relationships between the components of a computer system. It has been used by the Digital Equipment Corporation's manufacturing organization since January 1980. R1 is said to have sufficient knowledge of the configuration domain

thus it requires little search in order to configure a computer system. Reported errors were due to lack of information on new products [13]. R1 at the time of publication was in routine use. The acceptance of R1 in practice highlights the usefulness of expert systems technology.

PROSPECTOR

PROSPECTOR's [13] main purpose was the evaluation of the mineral potential of a geological region. It was used by geologists working in mineral exploration. Tasks included ore deposit identification and drilling site selection. In tests, PROSPECTOR was reported to have repeatedly produced results closely agreeing with those of geologist consultants [13]. PROSPECTOR's knowledge base contains models of different types of ore deposits. The success of the system in its domain justified efforts to extend the system's knowledge base.

DRILLING ADVISOR

DRILLING ADVISOR [23] was a rule-based system designed to diagnose oil drilling problems. It offered assistance to oil-well drillers on possible causes of problems encountered as well as solutions. It used backward chaining with certainty for inference, and frames as a knowledge representation scheme.

COMIX

COMIX [1], developed at Central Laboratories in New Zealand, is an expert system designed to give advice on the design of concrete mixes. COMIX is used in engineering and consulting disciplines, as well as by concrete technologists. Using the water over cement ratio, the system is able to calculate the amount of cement, course aggregate and sand required. Using mix designs based on the New Zealand code specification for concrete construction, the system refers the type of structure to the consistency and placement method [1; 21]. The system houses a rule and frame based knowledge base, with a resident authority as the information source. Development is on-going aimed at the extension of the knowledge base to include revisions of cement types and the strength factors thereof.

BETVAL

BETVAL [21], another expert system in the construction industry, is a rule-based system designed to give advice on the selection of ready-mix concrete at a job site. It facilitates the selection of the type of concrete ordered from a ready-mix concrete plant. BETVAL was developed by the Technical Research Center using Insight2+ [2] and an IBM PC/XT or AT computer. BETVAL is seen as a demonstration prototype, primarily used as a learning tool. It was noted at the time of publication that an extension of BETVALs knowledge would have to take place in order for BETVAL to be used as a production system [21].

Sports Injury Clinic

Sports Injury Clinic [45] is a web-based system that helps a user diagnose sports injuries. It presents a graphical user interface displaying different human body parts, and allows the user to point to problematic areas using the mouse cursor. The system gathers information from the user in an attempt to arrive at a conclusion. Like many other systems, Sport Injury Clinic offers a vast array of injuries, treatments and recommendations. The systems is limited in the sense that only muscular skeletal sports injuries are catered for. Besides this, the functionality and effectiveness of the system has lead to reviews of appraisal and great acceptance by experts in the medical field.

Expert System for Diagnosis of Pests and Diseases In Fruit Plants

Dewanto et al. [11] discusses the development of an expert system to diagnosis pests and diseases in fruit plants. The intended use of the system is to help users easily identify the type of disease in fruit plants. The design features primary components of an expert system, that is, a knowledge base, an inference engine, and user interface. The system employs backward chaining and utilizes a rule-based knowledge base to arrive at conclusions. Corvid Exsys [14] software, developed by Exsys company [1] was used to develop the system. The Exsys software tool features a rule editor with a visual interface of decision trees and an inference engine. It is possible to run applications developed using the Exsys

[1] http://www.exsys.com/

software tool online. To handle uncertainty, a formula is used to calculate confidence values in the system. During consultation, the traditional approach of interactively asking the user for values to be used in the rules is used. The user selects relevant answers to questions posed, based on observed symptoms. The system uses the answers provided together with information stored in the knowledge base to arrive at a diagnosis. Experiments showed that the output provided by the system is in accordance with the knowledge obtained from knowledge sources, hence proving the credibility of the developed expert system.

Summary

The discussion of systems above confirms the usefulness and relevance of expert systems in society. Expert systems in general have notable limits. These include the inability to reason based on intuition and common sense as an expert would. Furthermore, they have limited knowledge and inherently make it difficult to integrate knowledge from other domains. Often, the learning process of an expert system is not automated and thus requires intervention which is not always possible [19]. It is argued that providing tools that exploit new ways to represent knowledge for use in solving problems, and not the duplication of human behavior in all aspects, is the goal of expert systems research [13]. A vast array of systems exist that perform exceptionally well in their domains. Although problems exist that are believed to be holding back the improvement of such programs, research continues in an attempt to address these issues.

2.4 Attempts at Knowledge Base Automation

In an attempt to address problems in knowledge acquisition, research and development efforts have been directed at creating tools to enable the automated construction of knowledge bases.

EXPERT SYSTEM CREATOR

Expert System Creator [35] is one of the tools available for the development of expert systems. It is a portable development and integration environment for knowledge management, expert systems construction and validation, and database integration. Over

and above the tasks mentioned, Expert System Creator makes it possible to integrate the system with external projects. Representation of domain knowledge is achieved through the use of production rules, decision tables or classification trees. The Java programming language [4] is used to implement the tool. Expert System Creator uses CLIPS and JESS expert system shells for the reasoning process, and generates C/C++ and Java code for the decision tables and trees [35]. The use of such established tools contributes to the reliability of the output as well as the familiarity of the system to users. Pop et al.[35] note that the current status warrants further development, with respect to the representation, and support for automatic project documentation generation. Through the use of advanced widgets to support the knowledge acquisition phase and integration of fuzzy logic engines, enhancements in Expert System Creator are expected to address the current problems in knowledge representation. It is also noted that the integration of intelligent reporting tools will improve the performance of Expert System Creator. There is an apparent focus on integration with existing tools to achieve maximum efficiency. Although this may introduce complexities, when done correctly, significant improvement can be achieved. It is important to note the limitations in terms of flexibility and customizability that are inherent in robust systems like Expert System Creator. Providing a wide array of capabilities, Expert System Creator has been the tool of choice for various development endeavors [35], confirming its inherent strengths.

PROTEGE-2000

PROTEGE-2000, a knowledge management tool from Standford Medical Informatics [35], facilitates the construction of a domain ontology. In addition it customizes electronic knowledge acquisition forms used by experts and knowledge engineers during the knowledge acquisition process. Additionally, it encodes the knowledge into a database. However, in order to leverage the full power of the tool, it is necessary to integrate it with other existing knowledge acquisition tools.

KRITON

Diederich et al. [12] proposed a hybrid system, called KRITON, for automatic knowledge acquisition. KRITON [12; 7] combines cognitive science and AI methods to create knowledge bases using different representations. Declarative knowledge is acquired through

automated interview methods whereas protocol analysis is used for the acquisition of procedural knowledge. KRITON uses the knowledge acquired to construct and present an intermediate representation language that is subsequently used by frame, rule and constraint generators to build up the final executable knowledge base. A hybrid approach was taken in an attempt to address the knowledge-acquisition bottleneck. Through the use of hybrid knowledge acquisition tools, KRITON captures different kinds of knowledge thereby filling the gaps resulting from the limitations of using a single tool for representation.

KRITON uses three knowledge acquisition methods [12]. From AI, KRITON borrows the knowledge engineering strategy of interviews. KRITON relies on a dialogue between the expert and knowledge engineer to acquire declarative knowledge. From cognitive science, protocol analysis is used to acquire procedural knowledge. This involves the processing of texts acquired through the transcription of protocols of loud thinking during a problem solving process [12]. KRITON also makes use of incremental text analysis that enables acquisition of valuable knowledge from different sources. The acquired knowledge goes through an intermediate processing phase in which consistency checks and completion by inference takes place. The output of this phase is an intermediate representation language consisting of descriptive language for functional and physical objects and a propositional calculus. The introduction of an intermediate representation has proven to be an advantage in most implementations [7]. Intermediate output allows for monitoring and intervention where applicable. To complete the engineering phase, frame, rule and constraint generators build up the final representation using the intermediate representation language as input. The knowledge already in the knowledge base is used to guide subsequent elicitations thus aiding the incremental development and completion of the knowledge base.

KRITON presents an open, hybrid, and modular approach to the acquisition of declarative and procedural expert knowledge. An open system in this context provides facilities to enable extension and elaboration, while modularity guarantees ease of change and debugging. It is stated that the use of different acquisition tools and information from different sources makes the system hybrid. Hybrid systems have been noted to introduce inherent benefits [41]. Such features could arguably be the difference between an effective representation and an ineffective one. Diederich et al. [12] note however, the shortcomings of the KRITON system, namely, inaccurate and erroneous working of the protocol and text analysis. This has been addressed by the reliance on editors to aid employment and testing of the system. Furthermore, the lack of facilities to enable integration with other systems is a notable limitation. The emphasis on integration in the Expert System

Creator project [35] presented a flexible and extensible system. In this domain, the ability to extend a given tool is one that is crucial to the success of a project.

KELVIN

KELVIN [28] is a tool designed for the automated construction of knowledge bases. This is achieved through the processing of a text corpus and then refinement of a knowledge base about people, locations, and organizations. Unlike most systems in its domain, which facilitate the acquisition of knowledge through interviews, KELVIN makes use of natural language processing software. Knowledge is extracted from various sources with care. KELVIN uses asserted facts and information obtained from sources other than documents obtained from web-searches and various online collaboration communities. The knowledge extraction process includes detection of named entities, relation extraction, intra-document co-reference resolution and entity disambiguation [28]. KELVIN, as with KRITON [12], makes use of external tools to aid the construction of knowledge bases. These include tools to enable document annotations, add-on packages to enable annotations about identified entities, querying tools, as well as tools to enable the smooth integration with other tools. When tested, KELVIN showed evidence of learning. The system presented correct facts that it supposedly extracted from the unstructured knowledge source, suggesting a significant and acceptable level of success. The use text analysis and natural language processing is common in the knowledge engineering domain. Adoption is encouraged in a suitable environment and knowledge domain. Ongoing work on the KELVIN system as conveyed includes scaling to large corpora, detection of contradictions, expansion of inferences, exploitation of information extraction confidence and branching out to various genres of text.

Automatic Knowledge Acquisition Tool

Rafea et al. [36] presented a tool for the automatic acquisition of knowledge for irrigation and fertilization expert systems. The efficacy of the tool was measured based on the tool's flexibility, usability, accuracy, and exception handling. The tool's architecture comprises four main components: a knowledge elicitation module, a library, a knowledge base generator, and a verification knowledge base. The knowledge elicitation module is responsible for the creation, maintenance, and storage of information acquired from experts. Furthermore, it fetches information from the library and combines this with information elicited

from the user to create an intermediate knowledge structure. The library contains control and domain knowledge about the problem domain. The knowledge structure created by the elicitation module is used by the knowledge base generator to generate an executable system. This system is then verified by the verification knowledge base. The function of the verification knowledge base is to help experts test and verify the knowledge structure.

The modular structure of the system guarantees greater control. It enables the engineer to separate concerns, allowing focus to be placed on individual modules. Pre-defining problem solving methods and domain knowledge schema presents the domain experts with an easier task of filling in the gaps, which is achieved through automated interviews. This relieves developers of the burden of manually engineering the knowledge. Unlike most systems in its domain, which rely solely on elicited information, this system takes advantage of a separate differently structured base of knowledge. This arguably introduces improvements in the resulting representation. The presence of a verification knowledge base helps automate the testing phase, which in different circumstances would require continuous human intervention. None of the systems discussed above, that is, Expert Systems Creator, PROTEGE-2000, KRITON, and KELVIN incorporated automated verification facilities. The final output of the system is an executable knowledge base system. Rafea et al. [36] reported success of the proposed tool in the structuring of acquired knowledge by the use of relevant predefined domain knowledge to accelerate the knowledge base construction process. It is important to note that external effort would be required to construct and define the domain models, control knowledge, and domain ontologies contained in the library.

MORE

MORE [22], an intelligent knowledge acquisition tool is another attempt at addressing problems in knowledge acquisition. Aimed at assisting with the elicitation of knowledge from domain experts, MORE adds information acquired from experts through interviews to domain models of qualitative causal relations. The domain models are then used to generate diagnostic rules. MORE has functionality to elicit further information allowing the generation of a stronger set of diagnostic rules. MORE, which has seen use in several domains, presents a dynamic approach to knowledge elicitation and representation, which guarantees continuous relevancy of the knowledge base. Kahn et al. [22] demonstrate how MORE is used in parts of the drilling fluids domain. Furthermore, MORE has been used successfully to build systems to diagnose computer disk faults, network problems

and circuit board manufacturing problems. Successful use of MORE in these domains is argued to be an indication of the power of the strategies MORE employs [22]. Future work is directed at using MORE in the development of expert systems in various other domains. Although MORE is not as robust as some of the systems discussed, it presents itself as an solution to problems in its domain.

ICONKAT

ICONKAT [15] is an integrated knowledge acquisition system designed to facilitate the design, construction, testing, maintenance and explanation of knowledge bases. It comprises three subsystems: the knowledge elicitation subsystem, maintenance subsystem, and the explanation subsystem. The knowledge elicitation subsystem facilitates the construction of domain models from an expert's knowledge. The maintenance subsystem assists engineers and experts with testing the system's performance, refining the knowledge, and maintaining the system through the use of appropriate support tools. Concept maps and repertory grids [15] are utilized to provide various perspectives of the domain knowledge. The explanation subsystem makes use of constructed domain models as the foundation of the resulting consultant system's explanation capability. The use of subsystems enables the effective management of individual subsystems. Teams can work on different subsystems to improve performance. ICONKAT at the time of publication was in routine use for the purpose of designing and constructing systems for diagnosing first pass cardiac functional images [15]. ICONKAT, compared with the systems discussed, has plenty to offer. The robustness of the tool suggests exceptional performance, however, this is often at the cost of ease of use and implementation and integration.

ROGET

ROGET [6] is a knowledge-based system designed for the task of acquiring the conceptual structure of a diagnostic system. Through a dialogue with an expert, ROGET acquires the expert system's conceptual structure, which is then presented to developers to aid the design and development of the expert system. In addition, ROGET recommends system-building tools as well as identifying the scope of the resulting expert system. Although there is little emphasis on knowledge acquisition, ROGET presents excellent functionality in the domain of expert systems development.

Exsys Corvid Expert System Development Tool

The Exsys Company [2] has a reputation for providing expert system tools. Exsys provides a semi-automated tool for building and fielding interactive consultation systems online. With significant focus on ease of use and learning, the Exsys Corvid Expert system development tool aims to enable the conversion of domain expert knowledge and decision-making logic into a structured symbolic form capable of being dynamically executed by the Exsys inference engine during consultation. The development of Corvid has been assisted by organisations and business entities over a significant number of years, with the ultimate goal of developing a system that would allow fast and easy creation of simple and or complex systems online. The presence of online tutorials makes it easy for non-technical users to build systems over short periods of time. The Corvid system enables the development of expert systems by facilitating the capturing of decision-making logic of a domain expert, the generation of a suitable user interface based on the knowledge gathered, and finally, the integration of the system with other available IT resources. The uptake of Exsys services by users in various domains reflects potential credibility and reliability.

The aim of the discussion in this section was to highlight advances in the effort of exploiting new ways to effectively represent knowledge that can be used to solve problems. As noted, some of the systems excel under certain circumstances and fall short in others. This necessitates further research and development of such tools.

2.5 Tools and Techniques

It has been argued that the success in consultation of an expert system is highly dependent on the representation and organization of knowledge in the knowledge base [15]. Consequently, it is believed that the presence of a variety of tools and techniques to aid the knowledge representation process would result in the development of highly sophisticated systems that are capable of unprecedented levels of inference. It is not hard to argue for the necessity of research and development of such tools and techniques. These tools directly and indirectly aid the development of systems to enable effective knowledge acquisition and representation.

[2] http://www.exsys.com/

2.5.1 Client Interface

As discussed, the approach to knowledge elicitation has recently been the use of elicitation environments or systems. These present an interface that allows for a dialogue between a domain expert and a system. These allow the acquisition tools to facilitate automated interviews in an attempt to gather domain knowledge and the expert's expertise. A common characteristic of these systems is that they are web-based, which guarantees accessibility and mobility. Several tools exist that can be used in the development of interfaces that allowing for data input.

HTML

HTML (Hyper Text Markup Language) [18] is a markup language used for the creation of websites. An HTML document consists of HTML elements. A collection of these and plain text are used to describe the structure, content and semantics of a web document. HTML documents are read by a web browser, which composes them into visible structured web pages. It is possible to embed objects into web pages using HTML. Objects can include images, video, audio and other documents. Below is an example of an HTML document.

```
<!DOCTYPE html>
<html>
  <head>
    <title>I am a title</title>
  </head>
  <body>
    <p>Hello world!</p>
  </body>
</html>
```

The tags have special meaning attached to them, which is understood by web browsers. For example, the ¡!DOCTYPE html¿ tag denotes the document type to allow correct display of the web page.

As an open format, HTML is available for use without the need to acquire licenses. With HTML5 as the most recent version, HTML is supported by every web browser, consequently, it is used in the development of most applications, including mobile, desktop and

web applications. Additional benefits include its simplicity, ease of use and maintenance, and availability at no cost. One of the notable benefits is its ability to allow embedding of scripts to extend a websites functionality. HTML as a markup language presents itself as an excellent solution for creating web interfaces that can be adapted into a client interface for a knowledge acquisition tool.

JavaScript

JavaScript [32] is a programming language developed jointly by Netscape Communications Corporation and Mozilla Foundation. It is argued to be the most popular programming language, with JavaScript interpreters believed to be installed on every computing device capable of browsing the web. Its primary use is the creation of interactive effects in web browsers. It introduces interactivity within web pages and aids the development of web applications. JavaScript code is often embedded within HTML documents to extend the functionality thereof.

JavaScript code is interpreted, thus making it faster than traditionally compiled programming languages. However, modern implementations may be compiled in situations where a browser feels this will result in better performance. Like HTML, JavaScript is widely supported with most browsers capable of running JavaScript code. JavaScript is introduced in an HTML document using a special tag. An example is shown below.

```
<script type="text/javascript">
//JavaScript code
document.write("Hello World!")
</script>

<script src="/path-to-file/.js" name="name"></script>
```

The first example executes the JavaScript code enclosed between the script tags. In this case, a simple 'hello world' text is displayed on a web page. The second example imports a script from a defined location. The result will therefore be whatever the code in the input file does. Often invoked upon the click of a button or submission of a form, JavaScript

code enables the creation and manipulation of objects. For interactivity, JavaScript is arguably the best development tool available.

ASP.NET

ASP.NET [29] is a server side development framework developed by Microsoft for the creation of web sites, web applications and web services. Development models supported include web pages, model view controller, and web forms. It allows developers to create dynamic websites using a virtual interface.

Unlike notable alternatives, ASP.NET offers increased performance resulting from compiled code. Furthermore, it supports standard programming languages, hence eliminating the need to learn a new programming language. One other notable benefit of ASP.NET is its .NET framework foundation. This introduces .NET development tools that can be used to create Windows desktop applications as well as web applications. Developers can make use of the Visual Studio .NET tool. ASP.NET is widely used and favored as a framework for the development of web based interfaces. In addition, full technical support and in most cases, reliability is guaranteed, which may not be the case with open source alternatives.

A notable downside is the reliance of ASP.NET websites on ASP.NET compatible servers, that is, servers that support ASP.NET applications. At this point in time, ASP.NET is a viable and extremely capable tool for the development of interactive web interfaces.

Windows Forms

Windows Forms [31] is a smart client technology in the format of a graphical application programming interface (API) for the .NET Framework. It provides a platform for the development of robust applications with rich user interface elements by providing a set of managed libraries that simplify common applications. A form is a visual surface on which information is displayed to the user. These are created in a development environment like Visual Studio [3] by dragging and dropping user interface elements in a Windows Forms Designer. Forms can be used to request input from users, and communicate with remote computers over a network. Windows Forms applications are referred to as event-driven application. Actions are generated when the user interacts with the form, which

[3] http://msdn.microsoft.com/en-us/vstudio/aa718325.aspx

are then processed through the execution of supplied code. This code, often referred to as code-behind, can be written in any of the languages supported by the .NET framework. Windows Forms has the benefit of allowing the creation of robust applications with relative ease. Most of the code is managed allowing the developer to focus solely on the domain logic. However, as a Microsoft technology, Windows Forms applications function best on machines running Microsofts Windows operating system. This is seen by many as limitation and for this reason, attempts have been made to allow Windows form applications to run on non-Windows systems. As with ASP.NET, Windows Forms offers increased performance from compiled code. Furthermore, the underlying .NET framework introduces tools that can be used in conjunction with Windows Forms to create robust Windows applications.

2.5.2 Intermediate Representation

The benefits of an intermediate representation were highlighted in an earlier section. These include the ability to allow monitoring and management of the resulting representation. Several document formats exist that could allow the representation of somewhat intermediate data.

XML

XML [44] (eXtensible Markup Language) is a markup language designed for the store and transportation of data. Like HTML, XML is widely used in various domains and is favored for its simplicity, openness and extensibility. The structure of an XML representation is similar to that of HTML. As with HTML, it is possible to embed existing data within an XML representation. Markup languages present inherent benefits that render them excellent tools for storing data.

Comma Separated Values (CSV)

CSV, for Comma Separated Values, is a file format commonly used for the storage of delimited data in plain text format. Data elements are separated by comma characters with records terminated by newline characters. Although they are not used as much as notable alternatives, CSV files are used to exchange data between applications in various domains. Their structure permits the store of data representing sets or records. With

little effort, CSV can be used to effectively represent a variety of structured data sets. For this reason, it is an option worth considering.

Plain text

This refers to textual data in ASCII format which is said by most to be the most portable format due to availability of machine independent applications that support it. Characters supported include numbers and symbols. The format is limited and as a result does not support any type of formatting. Plain text file sizes are usually small with documents taking up less than half the size of rich text documents containing the same number of characters. The simple structure plain text enables files to be easily processed by different applications owing to their immunity to computer architecture incompatibilities. A file is generally considered plain text if it maintains a human-readable forms. For this reason, files containing markup or meta data are considered plain text. Ultimately, plain text files are excellent for storing data likely to be processed by different applications.

2.5.3 Translation

To transform acquired knowledge into an executable representation, some sort of translation has to take place. This translation process takes as input an intermediate representation and produces as output a final executable knowledge base system to be used by reasoning programs. This warrants the creation of such a translator.

Java

The Java programming language [4] is an object-oriented programming language designed by Sun Microsystems. Java, which is a high-level or imperative programming language enables the development of computer programs using English based commands. Java source code is compiled into Java bytecode, which is verified and interpreted for a native architecture. Java offers great power to developers because of its extensive libraries, which makes it a first choice for skilled and non-skilled developers developing applications in various domains. Java is easy to use and extremely reliable. The widespread use of the language as well as the ubiquity of devices running Java applications is an indication of this. In addition, Java is secure, concurrent and platform independent. This makes it an attractive choice in the development of robust applications.

C#

The C# programming language [30] is one of the programming languages supported by the .NET framework. Derived from the C programming language, it is a simple and powerful type-safe object-oriented language that enables the development of secure and robust applications that run on the .NET Framework. Common applications include Windows client applications, XML Web services, distributed components, client-server applications and more. C# is specified as a common language infrastructure (CLI) language. The language provides features that make developing solutions faster and easier; these include type-safety, garbage collection, simplified type declarations, versioning and scalability support, and many more. The language boasts syntax similar to that of C, C++ and Java programming languages, but by introducing unique features such as delegates and lambda expressions, it stands out as better a choice. Microsoft has implemented Visual C# from the C# language, which is commonly used for developing Windows applications using the Visual studio development environment. As highlighted earlier, the .NET Framework class library provides inherent including access to many operating system services and other well-designed classes, which significantly improve the development process. As one of the languages supported by the .NET framework, C# can be used to implement the different tiers of a Windows Forms application, that is, the presentation layer, domain layer, and data layer. Like Java, C# offers great power and flexibility to developers, which render it an attractive choice of a programming language for the development of robust applications in various domains, especially those intended to run on Microsofts Windows operating system.

Other Options

A compiler is a program designed to translate source code expressed in one language into another language. A compiler generator is a tool that enables the efficient development of compilers. Compiler generators are available that can be used to create a compiler suitable for the translation of an intermediate representation into a final executable representation. The effectiveness of compiler generators and the efficiency with which they achieve their task makes them attractive tools worth considering. Rule generators are an option to consider. These facilitate the development of rule based knowledge bases.

2.5.4 Testing

VP-Expert

Much AI research has focused on improving the already existing technology and finding better ways to exploit it. As a result, tools have been developed to aid the efficient development of expert systems in various domains. A shell, which allows for the building and maintenance of knowledge based applications [3], is a notable example. VP-Expert [17] is an expert system shell developed by Paperback Software International. It is a powerful and easy-to-use tool that has seen use in educational and commercial applications. It contains everything needed to run an expert system. This includes an inference engine, a rule editor and a user interface. The inference engine uses specially structured knowledge base files to facilitate consultation. The editor provides a means for creating and editing the rules in a knowledge base, while the user interface allows for the asking of questions, presentation of traces and explanations of the flow of execution where needed. The VP-Expert shell is capable of backward and forward chaining. Furthermore, it applies the concept of confidence factors. An expert system shell like VP-Expert introduces efficiencies during the development of expert systems. Use of such a tool rids a developer of the effort needed to create the main components of the expert system like the: Inference engine, working memory and user interface. Furthermore, it has proven easier to build systems in various domains, since the only component required is the problem domain information. Using an expert system shell in development of expert systems has proven to significantly eliminate overhead. However, the input of new knowledge across systems can be inflexible and sometimes complicated. It is important to note that the sophistication of a shell has a direct influence on the overall performance of the expert system, that is, an expert system running a shell with a poorly implemented algorithm, irrespective of the quality and extensibility of the knowledge base, can yield an undesired inferior result. The VP-Expert software package comes in two genres: a student version or educational version, and a commercial one. These versions differ in the extent or depth of functionality they provide with the former providing limited functionality and the latter featuring fully-fledged features that significantly increase productivity and performance. An expert system shell like VP-Expert in this case provides sufficient functionality and capabilities, making it a credible tool for consultation. Considering the two main internal components of an expert system, that is, the knowledge base, and the inference engine, there is evidence to suggest that the use of an expert system shell like VP-Expert can notably improve the development and testing of an expert system [10]. The shell as noted takes as input a knowledge base file. This can be fed into the system from an external

source or created within the system itself using built-in templates and procedures for data management. These files make up the knowledge base used by the inference engine during consultation. The shell employs built-in routines for reasoning with the rules and facts supplied in the knowledge base. The system is easy to use and does not require exhaustive configuration effort to operate. The engineer is effectively left with the job of creating and structuring the domain knowledge into a knowledge base. The VP-Expert shell has been designed to accept knowledge base files with knowledge structured as IF-THEN rules. The structure of a VP-Expert rule is shown below:

```
RULE rulename
IF antecedent
THEN consequent;
```

The system requires that every rule has a unique name, an antecedent, and a consequent. An example of rule is given below:

```
RULE Diagnosis_of_measles
IF   Diagnosis = measles
THEN Treatment = penicillin;
```

VP-Expert permits the use of variables and values in rules, as shown above. The above rule states that upon the discovery of measles in a diagnosis, we may conclude that the treatment is *penicillin* by assigning variable `Treatment` the value `penicillin`.

2.6 Summary

A brief overview of expert systems was presented. This highlighted the applications, elements and examples of expert systems. The relevance of knowledge representation within the context of expert systems development was conveyed. Emphasis was on knowledge acquisition, noted to be the bottleneck of expert systems development, and the various knowledge representation schemes available. Systems were discussed that aimed to address problems observed in the process of knowledge acquisition and representation. Their strengths, weakness and differences were noted. Finally, tools and techniques that could aid the development of such systems were explored.

Chapter 3

Knowledge Elicitation

This chapter discusses the design and implementation of the knowledge acquisition tool (KAT) with emphasis on the preliminary modules. An overview of the application is given, briefly introducing the different components of the application and steps that take place in order to get to the final output. Knowledge elicitation is discussed, that is, how the knowledge is elicited from the domain expert, and how this knowledge is stored. Next, a critical step in the knowledge base generation process is discussed, this is the intermediate knowledge generation step. The discussion touches on the tools used to enable the generation as well as the details on the output of this step.

3.1 Overview

The design of the KAT system, to some degree can be regarded as modular; modules with clearly defined functions make up the system. These work together in the background to produce the ultimate output of the KAT system, the knowledge base itself. The research objective is to present a way to automate the process of knowledge engineering when developing expert systems. This is achieved through the development of a system that will enable the automatic construction of executable knowledge bases. This is an attempt to provide a solution to problems inherent to the knowledge engineering process during the development of expert systems, which include time constraints and lack of communication between the parties involved, that is, the domain experts and the knowledge engineer.
The KAT system comprises three components or modules, which together are responsible for the overall working of the application. These modules enable the execution of the

critical processes, which in turn, enable the generation of a knowledge base. The processes are knowledge elicitation, intermediate knowledge generation, knowledge base generation, and verification. These processes and components are shown in Figure 3.1:

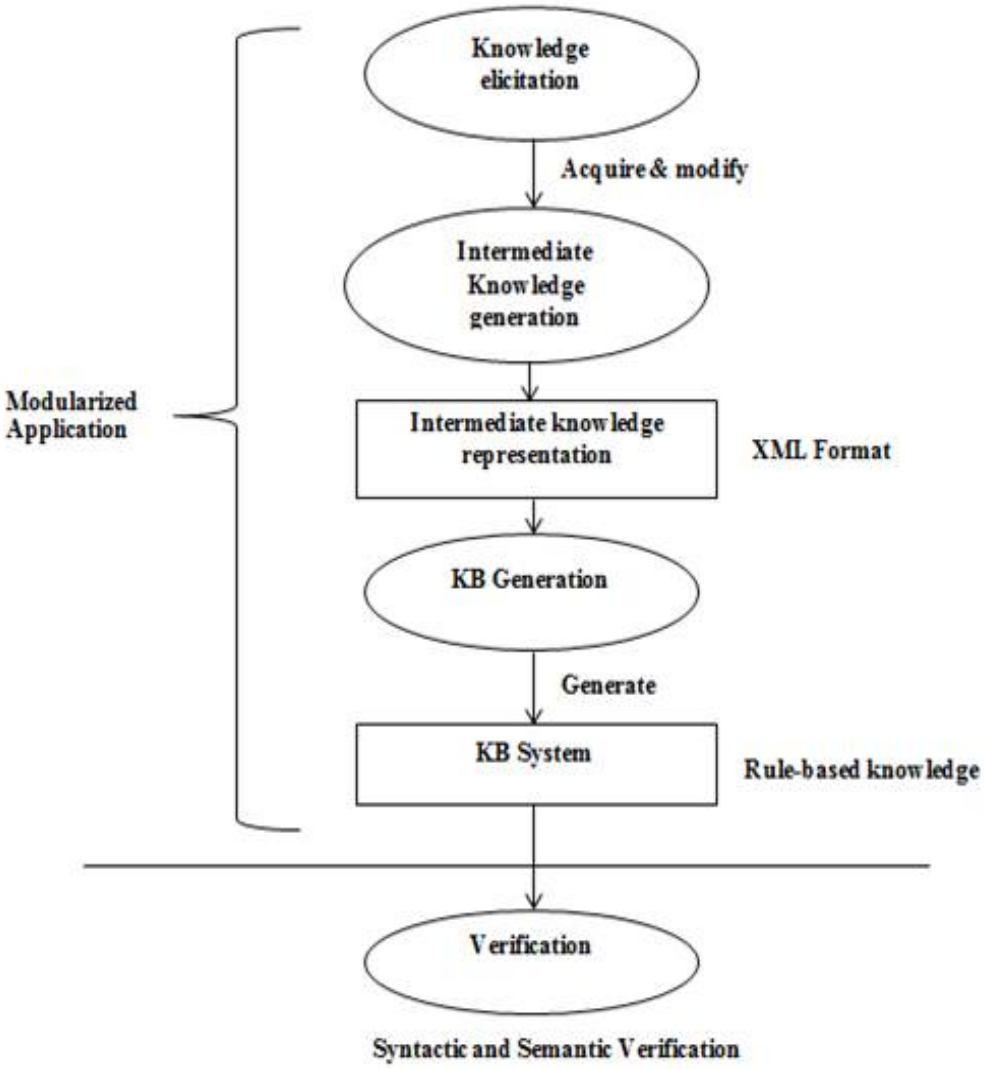

Figure 3.1: Steps, input, and outputs of the modules involved in the knowledge base generation process

The application features a user interface to allow the input of domain data and details about objects in a given domain. The user interface allows for the knowledge elicitation process to take place. The design of the interface facilitates what information is requested and therefore stored, and thus allows only relevant data to be entered, which ultimately results in the generation of an appropriate and relevant knowledge base. The application additionally features an intermediate knowledge generator, which takes as input the data entered by the user in the elicitation phase and converts this into a form appropriate for storage and transfer into external modules or applications. As the name suggests, this module enables the execution of the intermediate knowledge generation process, the output of which is an XML-based knowledge representation. The final module is the knowledge base generator or rule base generator, which is responsible for the generation of the final executable knowledge base. A VP-Expert expert system shell was used to carry out the evaluation of the system through the verification and validation of the constructed knowledge base.

3.2 Knowledge Elicitation

To demonstrate the functionality of the KAT system as a knowledge acquisition tool, the presence of knowledge is paramount. The problem domain can be argued to be of little significance as far as the research aim is concerned. The research objective, as noted, is concerned with the use of modern technologies to enable the automation of the knowledge engineering process. It was a design and implementation decision to select a problem domain that would present the least amount of difficulty in trying to achieve the research objective. With that said, sophisticated expert systems exist that focus on diagnostics, whether it is system diagnosis, or clinical diagnostics. For this reason, much research in recent years has focused on studying the use of diagnosis knowledge bases as well as extending the knowledge bases of such diagnostic knowledge-based systems, all this with the hope of achieving greater levels of inference. It is for the reasons mentioned above that the disease diagnosis problem domain was selected, specifically cattle disease diagnosis. The user interface and consequently the rest of the modules were developed with the problem domain in mind. It can be argued that this presents a limitation, however, it has sufficed for the demonstration of the intended functionality of the KAT tool.

3.2.1 Information source

The Cattle site [1] was used as the source of disease knowledge. The website provides comprehensive information on cattle diseases, which includes: disease names, typical symptoms, advice on prevention, and treatment information. Upon selection of the disease of interest on the website's webpage, information relevant to the disease is displayed. The KAT system was developed and tested using the knowledge acquired from the Cattle site. Listed below are items of information about a given disease that can be retrieved from the Cattle site.

- **Disease name**- This is the name of the disease, which sometimes includes the alternative names of the disease as most diseases can be identified using more than one name.
- **Causes** -This category details events, conditions or occurances that may result in an animal contracting the disease.
- **Treatment** - This category gives information on how an animal can be treated for the disease.
- **Prevention** - This explains how the disease can be prevented in the case that an animal has not yet contracted the disease.

The information provided by the website proved sufficient for the purpose of demonstrating the functionality of the KAT system.

3.2.2 Data elements of interest

To demonstrate the intended function of the system, a few data elements were required from the disease information collected from the Cattle site. When used appropriately, these data elements enable the effective generation of a knowledge base in the form of production rules, which can be used by an inference engine during consultation. These data elements are discussed below:

1. **Disease name** - This is used as the identifier for a given disease and is stored in the system as a string

[1] http://www.thecattlesite.com/

2. **Alternative name** - As noted, some diseases are identifiable by more than one name. This field stores the alternative names of a disease, provided one exists, in string format. If none is provided, it is assumed that the disease has only one name. The absence of this value does not affect rule generation in any way.

3. **Primary symptoms** - From the data source, symptoms have been identified that suggest the presence of a particular disease. These symptoms are grouped for the purpose of establishing priority. Primary symptoms are those that have to be present for a conclusion to be drawn, that is , they are the direct result of a pathogen. These are stored as a list of strings, as a given disease can have multiple primary symptoms.

4. **Seconday symptoms** - These are symptoms that present themselves in some cases and as a result are not always present.

5. **Treatment** - Stored as a sequence of strings, this is the suggested treatment that can be administered to treat an animal that has contracted the identified disease.

For each disease, the domain expert can use the graphical user interface provided to input data on the different data elements, and in doing so, populate an internally stored list of diseases.

3.2.3 Storing of acquired data

The KAT system stores elicited data in an intermediate form prior to subsequent processing by the next module. The information is initially stored in a human-readable form in a plaintext file. The storage of elicited data allows the domain expert to review the data as it is seen by the KAT system. If any errors or mistakes are identified, the domain expert is able to re-enter new correct data in the system to ensure that the system has correct information. The generation of a human-readable intermediate representation is done for the sole purpose of verifying the correctness of the elicited data. Shown below is an extract of the contents of the human-readable representation in plaintext.

```
Name: Anthrax
Othername: Anthrax
Primary symptoms: sudden death; fever; difficult breathing;difficulty
    swallowing; swelling of throat and neck
Secondary symptoms: muscle tremors; redness of mucous membranes;blood-
    stained discharge from nostrils, mouth and anus
```

```
Treatment: Due to the rapidity of the disease treatment is seldom possible;
    although high doses of penicillin have been effective in the later
    stages of some outbreaks.
```

Listing 3.1: Human-readable representation

The data corresponding to the data elements discussed is temporarily stored for verfication purposes. The contents of the generated plaintext file can be viewed using a simple text editor. This file, `HUMAN_READABLE.TXT`, is located in the program directory.

3.2.4 The user interface

The C# programming language was used to create the graphical user interface (GUI). As a powerful objected-oriented language supported by the .NET frame, C# stood out as a suitable language to use. The creation of the GUI was achieved by building a Windows Forms application intended to facilitate the knowledge elicitation process. During the elicitation process, the user is prompted for items of information that will help the rule generator to generate rules that can be used effectively by an inference engine to arrive at accurate diagnoses. Figure 3.2 shows a screenshot of the GUI with test data in the data fields.

The interface is made up of two sections or control groups: Disease Information, and Representation. These sections are used to group the different controls on the form and semantically separate the purpose of the user interface elements.

Disease Information

Several `Textfield` controls are housed in a `Groupbox` control to enable the user to enter disease information. These controls, namely, disease name, other name, primary symptoms, secondary symptoms, and treatment, accept data in the form of strings. It is worth noting that the design and inner workings of the KAT system requires that a semi-colon character (;) is used to separate symptoms in both the `Primary symptoms` and `Secondary symptoms` text fields, as shown in Figure 3.2 above. Failure to do so will result in an undesired generation output, which can compromise the credibility of the resulting knowledge base.

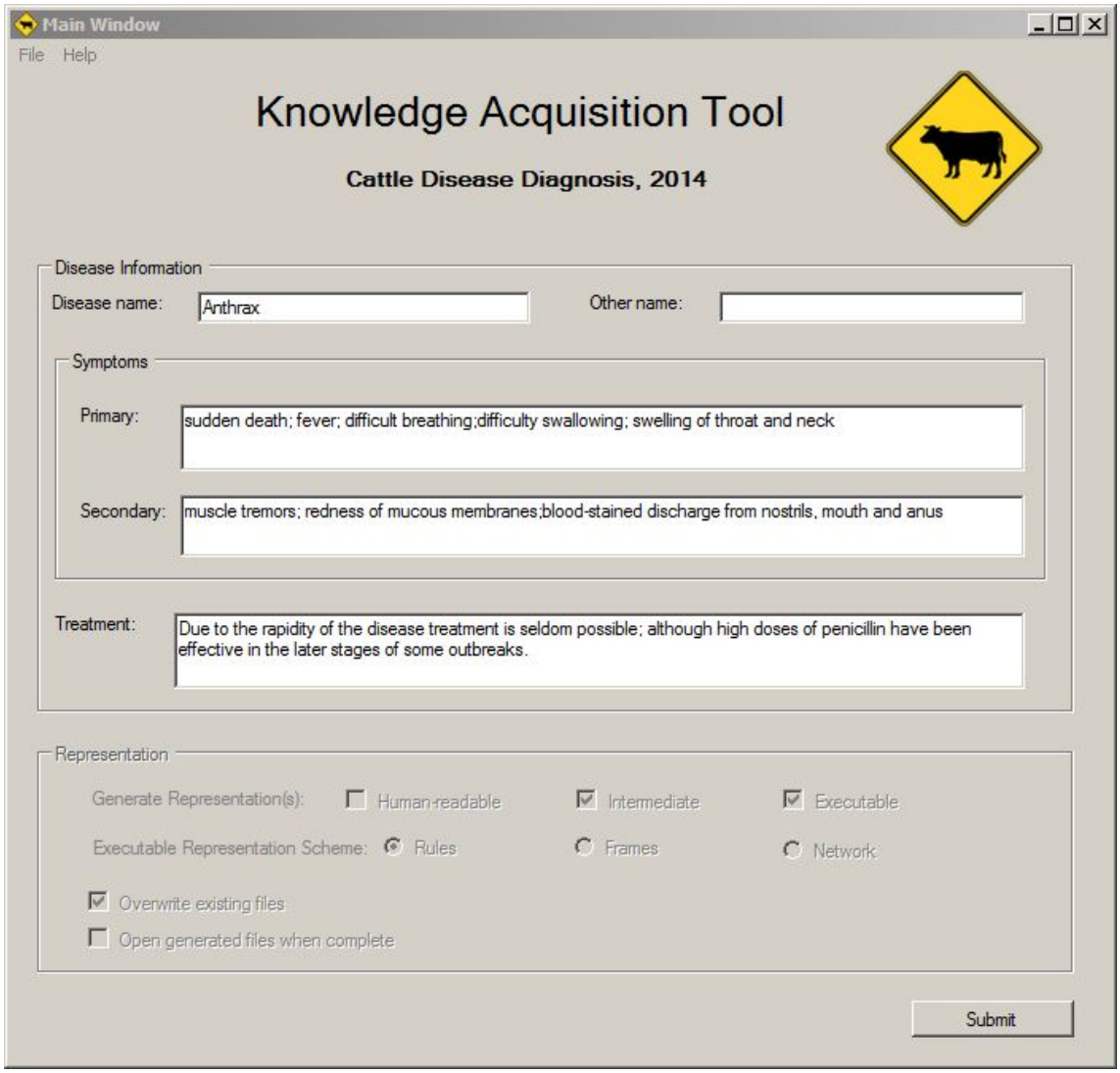

Figure 3.2: Screenshot of the KAT GUI

Representation

This section contains controls that enable the user to control how the application generates the various representations. The user interface control in this section can only be used after a disease has been added into the system. This is deliberate to prevent the "generation" of an empty knowledge base. The purpose of the different user interface controls are discussed below:

- **Generate Representation(s) controls**
  Three `Checkbox` controls allow the user to request the generation of a specific representation.

  - Human-reable - checking this enables the generation of a human-readable intermediate representation.
  - Intermediate - checking this enables the generation of an intermediate XML-based representation.
  - Executable - checking this enables the generation of an executable representation. This is enabled by default.

  Checking all three checkboxes consequently results in the generation of all three representations.

- **Executable Representation Scheme controls**
  Three `Radio button` controls allow the user to select the specific representation scheme for the resulting knowledge base.

  - Rules - checking this instructs the application to generate a rule-based knowledge base (KB).
  - Frames - checking this instructs the application to generate a frame-based KB.
  - Network - checking this instructs the application to generate a semantic network-based KB.

- **Overwrite existing files control**
  The KAT system allows for the incremental extension of the knowledge base. This can be achieved by appending new information to previously generated representations. By unchecking the `Overwrite existing files` checkbox, the user instructs the application to append the new information to existing representation files. Checking this control results in the overwriting of existing files.

- **Open generated files when complete control**
  This control allows the user to select whether the generated files should be opened automatically upon completion of the generation process. When checked, generated files are opened using the default text editor on the system running the application.

Adding a disease

In order to add information into the KAT system, the user will need to populate the relevant text fields on the form. To invoke the application to store the entered data, the user is required to click on the `Submit` button control located at the bottom right

corner of the form. When entered data has been successful stored, a dialog window is displayed confirming that the disease has been added. The user is then prompted to select whether he or she wishes to generate the knowledge base at that point. Figure 3.3 shows a screenshot of the dialog displayed when a disease has been successfully added into the KAT system.

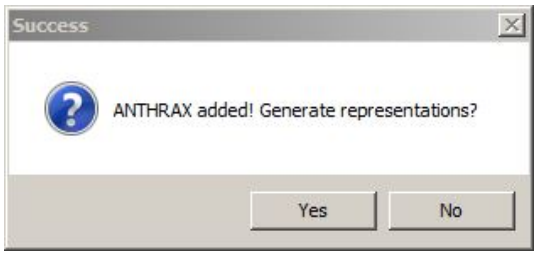

Figure 3.3: Screenshot of dialogue confirming successful disease entry

Selecting the `Yes` option invokes the generation of an executable knowledge base. It is important to note that the generation process only considers information that has been entered into the system up to this point. If the user selects the `No` option, the user is taken back to the main window, whereupon new data can be entered.

Input validation

To ensure appropriate use of the user interface, some input validation is incorporated into the user interface. Discussed below is the input validation that occurs in a common scenario.

- **Absence of critical required information**
  In the situation where a user attempts to add a disease by invoking the `Submit` method, the application checks the data fields to ensure that all critical data has been provided. If the tool detects that an item of critical information, like the disease name, is missing, an error message is displayed alerting the user to the omission of the data. A screenshot of an error message alert is shown in Figure 3.4.

The proposed user interface provides a means of communication between the domain expert and KAT system. Although limited in functionality and sophistication, the user interface has been shown to provide adequate functionality to allow for the demonstration of the tool's functionality considering the scope of the research problem.

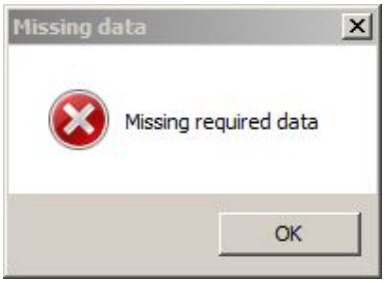

Figure 3.4: Screenshot of error dialogue alerting the user of missing data

3.3 Intermediate Knowledge Generation

The generation and presentation of an intermediate knowledge representation introduces benefits during the structuring and engineering of knowledge [7]. Similar to the KRITON system discussed in Chapter 2, the KAT system includes an intermediate knowledge generation step. The elicited knowledge is transformed into an intermediate representation that is used as input to the next module. The presence of an intermediate representation allows for the monitoring of the generation processes as well as the intervention by knowledge engineers and domain experts if needed.

3.3.1 XML Generetor

The XML generator uses the elicited information as input to generate an intermediate XML-based representation. This supports the integration of the knowledge elicitation module with the knowledge base generation module. Additionally, it allows for inter-system integration to take place.

XML representation

The intermediate knowledge representation, as noted, can be used as an input to an external module. XML is discussed in the literature survey as being an excellent means for data storage and transportation. These benefits are exploited in the KAT system. Using the intermediate knowledge representation, it is possible to integrate the KAT system with a different system. This is made possible by existence of a vast array of XML processing tools. The use of XML in this case presents the advantage of producing a structured intermediate representation of the data that allows for subsequent processing by disparate modules. The uses of an XML-based representation are expanded below.

- Temporary storage - XML is used to temporarily store data and information waiting to be processed by disparate systems or modules. When new data becomes available, this can be appended to the disease list stored in the .XML file. This data is then processed and used each time the rule generator is invoked.
- Monitoring - The resulting XML file can be analyzed to make sure that the overall generation process is progressing correctly. Mistakes can be identified and rectified earlier, thus ensuring that a credible and reliable knowledge base is generated.

XML elements

Data elements that make up a disease object are inserted into XML tags. The naming of the tags holds semantic relevance. Opening tags used for different data elements are listed below:

- `<diseaseList>` - encloses the list of the disease entered in the system.
- `<disease>` - encloses a disease object.
- `<name>` - encloses the name of a given disease.
- `<othername>` - encloses the alternative name of a given disease.
- `<symptoms>` - encloses the group or list of primary and secondary symptoms of a disease.
- `<primarySymptoms>` - encloses the list of primary symptoms belonging to a disease.
- `<psymptom>` - encloses a single primary symptom.
- `<secondarySymptoms>` - encloses the list of secondary symptoms belonging to a disease.
- `<ssymptom>` - encloses a single secondary symptom.
- `<treatment>` - encloses the treatment for the disease.

The tags listed above are carefully arranged to form the resulting XML file. For the purpose of demonstrating the usefulness of an intermediate representation during the engineering of knowledge, this short list of tags was found to suffice.

Invocation of the XML generator

The XML generator is invoked in the event that the user chooses to generate a KB in the form of one of the available representations in the KAT system.

- **Input**
  The generated file, as discussed earlier, contains a single disease list, which in turn contains a collection of disease objects. This disease list is populated using compiled disease data elicited from the user. It therefore is a precondition that the list of disease contains at least one element, that is, at least one disease should be entered into the system in order for generation to occur. This is a design and implementation decision to avoid generating an empty disease list. As a result, the method responsible for generating the intermediate XML-based representation accepts as input a non-empty list of diseases and a second argument, a variable the value of which specifies the mode of generation. These are discussed shortly.

- **Functional call**
  Shown below is an extract from the main program thread. This shows when and how the XML generator is invoked.

```
if (checkBox2.Checked || !checkBox2.Checked){
  //generate regardless
  Intermediate.Generate(diseases, mode);
  Integrator.Integrate("INTERMEDIATE.XML");
  generated = true;
}
```

Listing 3.2: Invoking the XML-generator

  The identifier `checkBox2` in this case represents the user interface control that allows the user to choose whether an intermediate generation should be generated. In the extract above, the XML generator is invoked regardless of the user's choice. This is an implementation decision that will be justified in a subsequent chapter. The function call `Intermediate.Generate(diseases, mode)` is of great interest in this discussion. This invokes the `Generate` method in the `Intermediate` class, which is responsible for generating the XML-based intermediate knowledge representation. The method accepts as input two parameters:

  - `List<Disease> disease` - a list of diseases populated with elicited disease data.

- `Boolean mode` - a boolean value used to identify the mode of generation. Two modes exist: overwrite and append. A `true` value instructs the system to overwrite the existing .XML file, while a `false` value instructs the system to append new information into the existing .XML file.

- **Procedure**

  The pseudo-code for the procedure responsible for generation is listed and explained in Listing 3.3.

```
N := LEN( Diseases )
DiseaseInfo := NULL
Output := NULL

FOR i := 1 TO N:
   DiseaseInfo  += ADD_TAG( Diseases[i].GetName() )
   DiseaseInfo  += ADD_TAG( Diseases[i].GetOtherName() )
   FOR j := 1 TO LEN( Primary_List ):
      DiseaseInfo  += ADD_TAG( Diseases[i].GetSymptom[j] )
   FOR k := 1 TO LEN( Secondary_List ):
      DiseaseInfo  += ADD_TAG( Diseases[i].GetSymptom[k] )
   DiseaseInfo  +=  ADD_TAG( Diseases[i].GetTreatment() )
   Output += DiseaseInfo
   DiseaseInfo := NULL
END
WRITE_TO_FILE( Output )
```

Listing 3.3: Generation of XML-based representation

To produce an XML-based disease list, the internally stored list of diseases is traversed. For each disease, the various data elements are extracted and enclosed between appropriate XML tags. Each data element is stored temporarily as a string. The final output is effectively the result of concatenating the contents of string variables in which the data elements are stored. An extract taken from the `Generate` method showing how a `disease name` element is stored, is shown below:

```
String disease_name = "\t\t\t<name>" + diseases[i].GetName().Trim() +
   "</name>\n";
```

Listing 3.4: Extraction of a disease name

In this case, a list of diseases is traversed, extracting the data element corresponding to the disease name. The value is enclosed between the `name` tag and then stored

in an appropriately named variable. Another extract showing how symptoms are extracted is shown below:

```
String p = "";
 for (int j = 0; j < diseases[i].GetPSize(); j++){
       p = p + "\t\t\t\t\t<psymptom>" + diseases[i].GetPSym(j).Trim()
            + "</psymptom>\n";
 }
```

Listing 3.5: Extraction of primary symptoms from a disease

Despite the existence of high-ranking sophisticated tools for XML conversion, the examples given above aim to present the fine detail and simplicity of the internal operations of the XML generator. Formatting is added to the output to make it human-readable. This is achieved by indenting elements appropriately.

Generation output

The output file contains a disease list containing a collection of disease objects. This file, `INTERMEDIATE.XML`, is located in the program directory and can be viewed using a text editor. Shown below is an example of an XML file containing one disease, generated by the XML generator.

```
<diseaseList>
<disease>
  <name>Anthrax</name>
        <othername>Anthrax</othername>
        <symptoms>
          <primarySymptoms>
           <psymptom>sudden death</psymptom>
           <psymptom>fever</psymptom>
           <psymptom>difficult breathing</psymptom>
           <psymptom>difficulty swallowing</psymptom>
           <psymptom>swelling of throat and neck</psymptom>
          </primarySymptoms>
          <secondarySymptoms>
           <ssymptom>muscle tremors</ssymptom>
           <ssymptom>redness of mucous membranes</ssymptom>
           <ssymptom>blood-stained discharge from nostrils, mouth and anus
               </ssymptom>
          </secondarySymptoms>
  </symptoms>
```

```
    <treatment>Due to the rapidity of the disease treatment is seldom
        possible, although high doses of penicillin have been effective in
        the later stages of some outbreaks.
    </treatment>
</disease>
```

Listing 3.6: Sample output of the XML generator

The generation of an intermediate representation is a critical step in the generation of a knowledge base system, as the output of this step is used in subsequent stages by the next module.

3.4 Summary

This chapter focused on describing the design and implementation of the preliminary modules of the KAT system. First, an overview of the system was given. The knowledge elicitation and intermediate knowledge generation modules were then discussed, focusing on the structure, inputs, operations, and outputs of these modules.

Chapter 4

Rule Generation

So far, discussion has focused on the preliminary steps needed to generate an executable knowledge base. This chapter discusses the design and implementation of the rule generator, which is responsible for the translation of facts and domain knowledge into an executable knowledge base. The discussion focuses on how rules are constructed, that is, the structure and syntax of the resulting knowledge base file. The preliminary steps discussed in the previous chapter lead up to the generation of a rule base that adheres to a specific syntax. Consequently, the rule base generation is the most critical step as the output of this step, that is, the KB, has a direct influence on the performance of an expert system during consultation.

4.0.1 Rule-based representation scheme

In the preceding chapter, different knowledge representation schemes were discussed. Discussion focused on the structure of knowledge items, that is, how knowledge is represented, the characteristics thereof, and the advantages of the different representation schemes. The output of the generation process is a rule-based knowledge representation. Knowledge is represented using a collection of IF-THEN rules. The use of rules significantly reduces the complexity of modifying and updating an existing knowledge base [25]. Additionally, rules reflect natural decision making, that is, they reflect how humans solve problems. Highly acclaimed knowledge-based systems like the MYCIN and R1 systems used rule-based knowledge bases[25; 27]. This confirms the effectiveness of rules in representing knowledge. For reasons mentioned above, a rule-based representation was chosen as the executable knowledge base generated by the KAT system. However, as provided for on the GUI, other representation forms could be implemented as future work.

4.0.2 Problem domain

The problem domain used to demonstrate the functionality of the KAT system is that of cattle disease diagnosis. The resulting knowledge base is intended to be used in an expert system designed to diagnose cattle diseases. This is achieved by querying the user, who is presumably a cattle owner, about different symptoms observed on an animal. The expert system must use the facts elicited from the user during consultation together with the knowledge stored in the knowledge base to try and reach a diagnosis. Upon reaching a diagnosis, the expert system should then recommend a treatment based on the diagnosis. It is worth stating that the selection of the problem domain in this case is not one that involved much thought. This follows the negligible influence or insignificance of the problem domain on the working of the system, that is, it can be argued that the problem domain has little significance on the working of the KAT system.

4.0.3 Syntax of rule base

The VP-Expert shell was used to execute the resulting knowledge base for validation purposes. How this was done, and the results thereof, are discussed in detail in a subsequent chapter. For now, it will suffice to present the structure or syntax of the rules required by the VP-Expert shell knowledge base.

A VP-Expert shell makes use of `.KBS` knowledge base file as the knowledge base. The file contains domain knowledge that the inference engine can execute during consultation. For this reason, it is possible to have a fully functional expert system in a given domain by simply plugging in a properly constructed knowledge base file. It is crucial that the structure of the resulting knowledge base file adheres to the syntax required by the inference engine. A VP-Expert knowledge base is made up of three sections or statement blocks: the Action statements block, production rules, and the Query statements block. The different functions and significance of these blocks are explained below:

Actions block

The Actions block contains statements that control the execution of the inference engine. These are executed in order of appearance. Examples of such statements include the `FIND` statement,and `DISPLAY` statement.

`FIND variable` - This statement invokes the inference to begin inference by consulting the knowledge base until a value for `variable` is found. Consequently, every knowledge base file should contain a `FIND` statement as this activates the inference engine.

`DISPLAY "text"` - This statement allows text to be displayed on the screen. Text should be enclosed in double quotation makes. Inserting the tilde character (~) at the end of statement has the result of extending the duration the text is displayed on the screen. VP-Expert allows the value of a variable to be displayed as text by including the variable name enclosed in curly brackets, and preceded by the hash character (#) in a `DISPLAY` statement.

Production rules

The production rules section contains the production rules that are executed by the inference engine during inference. These effectively represent the domain knowledge and as a result have a direct impact on the performance of the expert system. Expressed as IF-THEN statements, the rules are executed on demand during the process of backward chaining. The design of the VP-Expert shell permits the inclusion of special operators in the rules. These include mathematical, relational, and logical operators. An example of a VP-Expert knowledge base rule extracted from a disease diagnosis knowledge base is shown below:

```
Rule 1
IF  sudden_death = observed
AND fever = observed
AND difficult_breathing = observed
AND difficulty_swallowing = observed
AND swelling_of_throat_and_neck = observed
THEN Disease = Anthrax;
```

It can be said that the structure of the rule is simple and human-readable. The rule above simple states that if the listed symptoms, that is, sudden death, fever, difficulty breathing, difficulty swallowing and swelling of the throat and neck, are observed, then the conclusion should be reached linking the diagnosis to the disease associated with the listed symptoms, in this case Anthrax.

Query block

The query block contains statements that allow the shell to prompt the user for values of variables that do not appear in the consequent of some rule. This is needed because there is virtually no other way the inference engine can determine the value. As a result, this is achieved by the use of special statements, namely, the `ASK` statement and the `CHOICES` statement.

`ASK variable: "prompt";` - Prompts the user for the value of `variable`. It is crucial that the prompt be as informative as possible so as to avoid any confusion on the part of the user with regard to what information is needed and how this should be entered into the system.

`CHOICES variable: list of values;` - These statements present a list of possible values assignable to the variable . It is important to note that the user can only answer the question asked in the Ask statement with the values supplied in the list.

Complete knowledge base

Using the statement blocks discussed above, a functional and syntactically correct knowledge base file can be constructed. The code listing below is a typical arrangement of the statement blocks in a knowledge base file.

```
ACTIONS
Production Rules
Query statements
```

Listing 4.1: Arrangment of different blocks

```
ACTIONS

  DISPLAY "Welcome to the expert system
  Press any key to begin.~"
  FIND Today
  DISPLAY "Today is {#Today}.";

RULE 1
  IF   Day = monday
  THEN Today = workday;
```

```
ASK Day : "What day is it?";
CHOICES Day : sunday, monday;
```

Listing 4.2: Example of a simple knowledge base file

In the example given in Listing 4.8, the result of invoking the inference engine using the given knowledge base file, provided that when prompted, the user selects `monday` as the answer to the question posed, will be to display the statement: *Today is workday.* . This is because evaluating `Rule 1` as specified above, results in the assignment: `Today = workday`, owing to the antecedent of the rule yielding a `true` value. The above example presents the structure of a VP-Expert knowledge base and the basics of how it is executed.

4.0.4 Automated generation

The successful automation of the generation of production rules is the measure of success as far as the research aim is concerned. It remains primary, however, that the rules are meaningful and capable of guiding the inference step toward a valid and accurate conclusion. This requires the careful formulation of individual rules, and the meticulous implementation of the algorithms thereof. There is need to emphasize that the resulting knowledge base should be executable by a suitable inference engine. This will require the careful generation of various syntactically correct control statements applicable to the knowledge base linked to the executing inference engine.

Rule types

The KAT system implements several types of rules when constructing a knowledge base that can be executed by a suitable inference engine. These rules are sufficient to guide the inference step well enough to reach reasonable conclusions. Explained below are the algorithms for basic rule generation. Additionally, the pseudo-code for these rule algorithms is supplied. It is important to note that although several types of rules are generated during the knowledge base generation process, the discussion below focuses on key rules that form the core of the knowledge base.

```
RULE Type 1
Diagnosis = NULL
```

```
IF All main symptoms of Disease X are present:
THEN Diagnosis =  Disease X
```

Listing 4.3: Rule type 1

The rule above states that after querying the user for the observed symptoms in an animal, if all the main symptoms of a disease are reported present, then it should be concluded that the animal has contracted the disease associated with the reported symptoms.

```
RULE Type 2
IF At least one secondary symptom of Disease X is present:
THEN Disease = probable
```

Listing 4.4: Rule type 2

The rule above states that after querying the user for observed symptoms, if at least one secondary symptom of the disease has been observed, it should be noted that the disease is probable. This `probable` status is used in a subsequent rule to reach a defined conclusion.

```
RULE Type 3
IF Disease X is probable:
AND At least one primary symptoms of Disease X has been observed
THEN Diagnosis = Disease X
```

Listing 4.5: Rule type 3

The rule above states that if a disease is probable and at least one of the main symptoms belonging to the disease has been observed, a conclusion should be reached confirming the diagnosis.

```
RULE Type 4
IF Diagnosis = Disease X:
THEN Recommend Treatment for Disease X
```

Listing 4.6: Rule type 4

Rule 4 simply states that if a diagnosis has been reached, the expert system should proceed to recommend the assigned treatment for the disease identified.

It is believed that the four rule types listed above have the potential to guide an inference engine toward valid conclusions or diagnoses. Rules not discussed above serve to complement the rules discussed here and as a result allow for the correct execution of the overall knowledge base, which in turn enables the inference engine to arrive at a diagnosis.

Invocation of rule generator

- **Input**
  As with the previously discussed XML generator, the rule generator requires that at least one disease exists in the data store. Consequently the rule generator accepts a non-empty disease list in XML format, which it uses to generate rules. The list of diseases can come from an external source like a separate program, or from an internal source like a separate module within the KAT system. This is a design decision to achieve modularity and integration of the KAT system with different applications.

- **Function call**
  The rule generator is invoked from within the user interface thread when the user instructs the system to start the generation process. The code listing showing how the rule generator is invoked is given in Listing 4.13.

```
if (checkBox3.Checked){
  //final representation
  RuleGenerator.Generate();
  generated = true;
}
```

Listing 4.7: Invoking of the rule generator

  The generator is set to execute only when it has been specified to do so. This is achieved through the checking of a `Checkbox` control on the GUI. In this case, `checkbox3` is the control that enables the user to make this decision. From the listing above, the rule generator is executed when the control enabling the generation of an executable knowledge base is checked. Consequently, if this control is unchecked, the rule generator is not invoked and thus rules are not generated.
  It is important to note that when calling the rule generator, unlike with the XML generator, it is not necessary to specify the mode of generation. This is because each time the rule generator is invoked, a new knowledge base is constructed using newly acquired disease knowledge together with previously acquired knowledge. This recompilation of old knowledge to construct new rules is made possible by

the generation of the intermediate knowledge representation. This necessitates the invocation of the XML generator by default each time the user chooses to generate a knowledge base. Shown below is an extract showing how a new disease list is created from a newly generated XML-based knowledge file.

```
public static void Generate(){
  List<Disease> diseases = Integrator.Integrate(path);
  Generate(diseases);
}
```

Listing 4.8: Method definition for Generate method

The code listing above defines an overloaded method that invokes the rule generator by calling the `Generate(List<Disease> diseases)` method, which accepts as input a single argument:

- `List<Disease> diseases` - a list of disease objects.

A new disease list is created from the contents of the intermediate knowledge representation file, the location of which is specified by the `path` variable. The purpose of the `Integrate(String path)` method here is to aid the integration of the intermediate knowledge generation and knowledge base generation modules in the KAT system. This method simply reads in the contents of the previously generated XML-based file from the specified location and compiles these into a list, which is later passed to the `Generate(List<Disease> diseases)` method. This in turn, is responsible for the generation of rules. It is therefore crucial that the `path` variable specifies a valid location where a correctly structured XML-based file can be found.

- **Procedure**
  For the generation of a complete knowledge base to be possible, three groups or blocks of statements have to be generated: the Action block statements resident to a VP-Expert knowledge base file, production rules to guide the inference step, and finaly, Query block statements.

  I **Generating Action block statements**
  As discussed earlier, VP-Expert requires the presence of statements to control the execution of the inference engine. These statements are predefined and as a result do not change much. The generated statements do however need to be specific to the knowledge base at hand, that is, the statements must relate to the domain knowledge semantically. This effectively implies that the

DISPLAY statement is succeeded by informative text specific to the problem domain. Furthermore, the FIND statement should be succeeded by a variable the value of which is assigned in the consequent of some rule in the knowledge base. The code listing of the definition of the method responsible for generation of Action block statements is given in Listing 4.15.

```
public static String ShowActionsBlack(List<Disease> xs){
  String actionsBlock = "", indent = "\t ";

  actionsBlock += "\nACTIONS\n" + indent + "DISPLAY \" Disease
     diagnosis\"\n";
  actionsBlock += indent + "NumProbable = 0\n";
  for(int i=0; i < xs.Count; i++)
        actionsBlock+= indent + varDiseases[i] + " = 0\n";
  actionsBlock += indent + "FIND Treatment\n";
  actionsBlock += indent + "DISPLAY \"Conclusion has been reached
     :\"\n";
  actionsBlock += indent + "DISPLAY \"Disease: {#Disease}~\"\n";
  actionsBlock += indent + "DISPLAY \"Treatment: {#Treatment}~\";\
     n\n";

  return actionsBlock;
}
```

Listing 4.9: Method definition for ShowActionsBlock method

The generation of Action block statements, unlike production rules, depends somewhat on the number of diseases stored in the system. For the purpose of improving the performance of the expert system during consultation, the disease list is used to aid the generation of a list of variables used to identify each disease in the knowledge base. From the code listing above, it can be seen that the generation of Action block statements simply requires the stringing together of appropriate keywords together with relevant textual information, which to some degree relates to the problem domain.

II **Generating production rules**

Production rules are generated using the rule algorithms discussed in subsection 4.4.1. The generation of production rules constitutes a large portion of the overall work done by the rule generator in the KAT system. As a result, production rules utilize the most amount of space in a knowledge base file. Despite that, it is worth noting that the size of the resulting knowledge base depends heavily on the number of diseases stored in the KAT system. This is

because the elicited disease knowledge is used to assemble the different parts of generated rules. In the discussion of the structure of IF-THEN rules in Chapter 2, it was emphasized that an individual rule needs to have an antecedent or condition, and a consequent or conclusion. Disease attributes like the disease name and symptoms are intelligently used as antecedents and consequents of different rules. Listed below are steps for the process of constructing a rule in a KAT-generated knowledge base.

Steps for rule construction:

For each disease in the disease list

1. Create and initialize a rule placeholder Ri
2. Append rule name or number to Ri
3. Append antecedent keyword (IF) to Ri
4. Append premise
5. Append logical operators if any (AND/OR)
6. Append premise
7. Repeat steps 5 to 6 if more premises
8. Append consequent keyword (THEN) to Ri
9. Append conclusions
10. Specify and append ELSE clause if needed
11. Append semicolon character at end (;) to close off rule

The eleven steps listed above implement the rule algorithms as discussed in subsection 4.4.1, thereby enabling the automated construction of rules. Brief explanations of the steps listed above are given below. For this discussion, the construction of a rule of `Rule type 1` is assumed.

Explanations:

Step 1: A placeholder or variable for the rule is created and initialized. Initially, the variable contains nothing but an empty string. On completion of the execution of the rule algorithm, this variable will hold the entire rule.

Step 2: The VP-Expert shell requires that every rule has a unique name or identifier. In the KAT system, sequential rule numbers are used as rule identifiers. This is achieved by incrementing a rule counter each time a rule is created. If the particular rule is the first to be generated, it will

consequently have the identifier: `1`.

Step 3: The VP-Expert shell, in addition requires that every rule contains predefined keywords to enable the identification of rule parts by the inference engine. The `IF` keyword is one of such keywords. As emphasized in previous sections, this has to precede the first premise of a rule.

Step 4: As discussed, `Rule type 1` checks for the presence of all the primary symptoms of a disease during a consultation, if present, the diagnosis is concluded. In this particular case, symptoms of a particular disease are combined with relational and mathematical operators, variables, and values to form premises. These are in turn combined to form the antecedent of a rule so as to allow the inference engine to narrow down the list of possible diseases by querying the user for observed symptoms.

Step 5: Logical operators are used to combine the premises of a rule.

Step 6: In principle, a logical operator should be succeeded by a premise.

Step 7: A VP-Expert shell knowledge base rule can contain a maximum of twenty premises. This limit is exploited in the KAT system.

Step 8: To mark the beginning of a consequent, VP-Expert uses the `THEN` keyword.

Step 9: The THEN part of a VP-Expert knowledge base rule can contain multiple conclusions. These usually take the form of an assignment like: `variable = value`.

Step 10: The VP-Expert shell permits the specification of an ELSE clause. This, if specified, is processed when the evaluation of a rule's antecedent yields a false value thereby restricting the execution of the primary consequent in the THEN part of the rule.

Step 11: VP-Expert uses a semicolon character to mark the end of a rule.

The remainder of the rules that make up the knowledge base are constructed in a similar manner. The semantics vary depending on the function or purpose of the rule. As highlighted earlier, the size of the resulting knowledge base depends on the number of diseases stored in the system. The same applies to some rules, that is, the structure and hence the complexity of a rule depends on the size and contents of the internally stored disease list. Listed below is a code snippet of a method that can be used to construct a rule. This has been included to supplement the discussion above.

```
//Rule A logic:  if all main symptoms are present, then conclude
    diagnosis
```

```
//-------------------------------
String final_rule_a = "", rule_a = "";
for (int a = 0; a < diseases.Count(); a++){
  rule_a = "Rule " + (ruleNumber++) + "\n" + "\tIF " +
     ShowPrimarySymptoms(diseases[a], "AND") + "\n\tTHEN Disease =
      " + diseases[a].GetName() + ";\n";
  final_rule_a += rule_a + "\n";
}
```

Listing 4.10: Construction of a rule of Rule Type 1

The production rules block is formed as a result of successfully stringing together the generated rules in the order in which they are generated. In this case, the same rule is generated for each disease, that is, the number of rules that implement the logic of the rule above, will be equal to the number of diseases stored. Simple but meaningful production rules are generated using the approach discussed above. It is worth stating that the design and implementation of more sophisticated rule algorithms will more often than not result in the generation of a more robust and knowledgeable rule base. Rules have been deliberately kept simple to allow the simplified demonstration of the proof of concept system.

III **Generating Query block statements**

The Query block as highlighted allows the inference engine to query the user for values not known to the inference system. Similar to the generation of Action block statements, the generation of the Query block statements depends heavily on the contents of the disease list. This is because the query statements generated make use of the different symptoms associated with stored diseases. Furthermore, similar to the generation of the Action block statements, the generation of Query block statements requires the linking of appropriate keywords with disease symptoms and appropriate prompt messages. The result of this is the formation of a standard query statement that can be recognized by the shell's inference engine. Listing 4.17 gives the code for the method responsible for this.

```
public static string ShowQueryBlock(List<Disease> xs){
  String queryBlock = "\n\n";
  String[] wording = { "Any signs of ", "Any ", "Is there " };
  Random rnd = new Random(); int index = 0;
  foreach (Disease d in xs){
    for (int i = 0; i < d.GetPSize(); i++){
          index = rnd.Next(3);
```

```
                queryBlock += "ASK " + Delimit(d.GetPSym(i).Trim().
                    ToLower(), '_') + " : \"" + wording[index]  + d.
                    GetPSym(i).Trim().ToLower() + "?\";\n";
                queryBlock += "CHOICES " + Delimit(d.GetPSym(i).Trim().
                    ToLower(), '_') + " : Yes, No;\n";
            }
            for (int j = 0; j < d.GetSSize(); j++){
                index = rnd.Next(3);
                queryBlock += "ASK " + Delimit(d.GetSSym(j).Trim().
                    ToLower(), '_') + " : \""+ wording[index] + d.GetSSym
                    (j).Trim().ToLower() + "?\";\n";
                queryBlock += "CHOICES " + Delimit(d.GetSSym(j).Trim().
                    ToLower(), '_') + " : Yes, No;\n";
            }
        }
        return queryBlock;
    }
```

Listing 4.11: Method definition for showQueryBlock method

Query statements come in pairs: an ASK statement, and a CHOICES statement. In the KAT system, each pair corresponds to a single symptom belonging to a particular disease. As a result, the generation of query statements requires accessing and extracting every single symptom of each and every disease in the disease list and combining this with textual parts to form syntactically correct query statements, as done in the method for which the code is given in Listing 4.17.

The result of combining the different blocks discussed above is the formation of an executable VP-Expert-recognized knowledge base. It is paramount that the arrangement of the different blocks is correct and adheres to the VP-Expert specification as described in subsection 4.3.4.

Generation output

The result of the successful invocation of the rule generator is the generation of a new knowledge base. This knowledge base is the final output of the KAT system, which can be loaded into the VP-Expert shell and therefore consulted during inference. The file, EXECUTABLE.KBS, is located in the program directory and like the other previously generated files, it can be viewed using a text editor. The results obtained from an attempt to execute the knowledge base are discussed in Chapter 4.

4.1 Summary

This chapter discussed the design and implementation of the rule generator. The representation scheme, problem domain, and the syntax of the generated knowledge base was presented. Finally, the construction of rule types and different knowledge parts was discussed.

Chapter 5

Evaluation of the Knowledge Base

In this chapter, the evaluation of the KAT system is discussed. Focus is on the verification and validation of the generated output, that is, how the established objectives have been met. The validity of the knowledge base is tested against a hypothetical inference engine and subsequently against an actual expert system shell. The limitations identified during evaluation are discussed. Finally, a discussion on the overall evaluation phase is given. As discussed in Chapter 2, evaluation of the resulting knowledge is a crucial step in the knowledge base construction process. This is done in order to verify and validate the output of the KAT system. Evaluating the generated knowledge base involves ensuring that the objectives set at the beginning of the project were met. As emphasized, the project aimed to present an efficient approach to the knowledge representation process in the development of expert systems by automating the knowledge engineering process. This has been achieved by developing a tool to enable the automated construction of a rule-based knowledge base the design and implementation of which was discussed in the previous two chapters.

5.1 Success criteria

In an attempt to measure the success of the project, success criteria were put in place. These aim to present the extent to which the developed system fares in representing knowledge.

5.1.1 Representing knowledge as rules

The construction of a knowledge base involves converting the expertise of human experts into a symbolic representation that can be executed by a suitable inference engine. This process usually involves collaboration between one or more human experts and a knowledge engineer. The human experts provide their expertise, which is intelligently programmed into a computer program by the knowledge engineer. In a rule-based system, this is the process through which facts are converted into knowledge represented as rules. The KAT system provides functionality to enable this conversion. The analysis of individual rules in the knowledge base can reveal that expert knowledge, as extracted from a credible source, has been successfully represented as rules. Listing 5.1 shows a rule from the knowledge base as generated by the KAT system's rule generator. The rule above states that if any of the symptoms listed have been observed, Blackleg disease should be concluded as the diagnosis.

```
Rule Blackleg
        IF   Lameness Loss of appetite
        AND  Rapid breathing
        AND  Fever
        AND  Unwillingness to move
        THEN Disease = Blackleg;
```

Listing 5.1: Example rule in the generated knowledge base

According to the information contained on The Cattle Site [1] on the Blackleg disease, symptoms of the disease include: lameness, loss of appetite, rapid breathing, fever, and an unwillingness to move. This suggests that the presence of the listed symptoms in an animal could in turn suggest the possibility of a *Blackleg* diagnosis. This knowledge has been successfully represented as an executable rule above. This serves as evidence of the ability of the developed tool to effectively represent knowledge as rules. Over and above, the generated knowledge representation was found to be in accordance with the knowledge obtained from the knowledge source.

5.1.2 Reaching valid conclusions

The development of an expert system is successful if the resulting consultant system effectively represents the expert knowledge that has been programmed into it. This requires

[1] http://www.thecattlesite.com/diseaseinfo/187/blackleg

that the system successfully transfers expert knowledge to a user during consultation. Consequently, the execution of the system should result in the inference engine reaching a valid conclusion, that is, conclusions reached should be in accordance with the conclusion reachable from consulting the knowledge obtained from knowledge sources. This can be achieved by constructing a complete and extensive knowledge base to enable the inference engine to draw an acceptable conclusion each time. This ability is primary as it has a direct influence on the performance of the overall expert system. Considering the scope of the expert knowledge as contained in the knowledge source and the scope of the research project, the resulting knowledge base as generated by KAT system was found to allow for valid deductions. This is demonstrated in subsequent sections.

5.1.3 Execution by suitable inference engine

Run-time evaluation is used in an attempt to demonstrate the knowledge base's potential for execution by an inference engine. Chau et al. [10] demonstrated that while effective structuring of knowledge bases is crucial for expert system design, reliable evaluation must take place for the acceptance of artificial intelligence technology. Evaluation is described to have two phases: verification and validation. Verification involves making sure the specifications put in place are met by the implemented system, whereas validation involves ensuring that the system correctly performs the intended purpose, that is, it is concerned with the accuracy and correctness of the end result. Chau et al. [10] additionally highlighted the importance of the mentioned steps and also the need to engage in such. The common approach has been to perform multiple tests with knowledge engineers and domain experts, which at times has proven impractical due to time constraints and lack of expert resources. Over and above, it was found that the approach did not guarantee the discovery of all errors, thus further complicating the process. A run-time evaluation was proposed as a supposedly efficient approach; this can be done by using a specially designed inference engine to execute the knowledge base, or by using a carefully selected expert systems shell. This was the approach taken to verify and validate the KB developed by the KAT system.

5.2 Manual evaluation

As emphasized, the project mainly focused on the automated construction of an executable knowledge base and not necessarily on the other components that make up an expert

system. Therefore, to argue for the possibility of automated knowledge base construction, the introduction of a hypothetical inference engine is justified. This is done to show the potential for execution of the automatically generated knowledge base. Figure 5.1 shows the structure of the proposed inference engine.

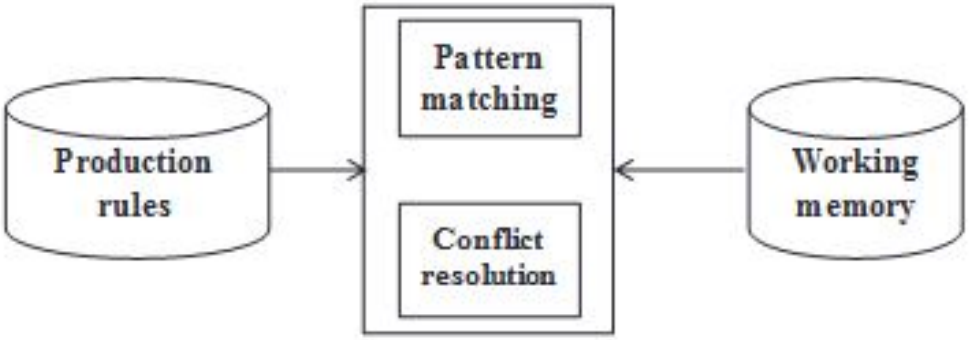

Figure 5.1: Structure of proposed inference engine

The proposed engine features a rule base of production rules, a working memory, and internal modules that enable inference to take place. These include pattern matching and conflict resolution. The engine uses pattern matching to repeatedly match facts and data against the production rules to draw conclusions. This is the basic functionality of an inference engine. In the working memory, a hypothetical list is stored. In this list, the system stores probable diseases depending on the symptoms reported at any given point in time. Production rules govern the insertion of diseases in the intermediate list. Once all observed symptoms have been reported and entered into the system, the system consults the list of diseases and employs conflict resolution strategies in an attempt to narrow down the intermediate list of probable diseases to a single element: a single probable disease. It is worth stating that the correct functionality of the different components shown in Figure 5.1 is critical for the effectiveness of the proposed inference engine. To demonstrate this, a simple case is discussed below.

Given a list of three diseases with associated symptoms:

1. **Disease name** - Anaplamosis
   Primary symptoms - Amenia, fever, weight loss, breathlessness
   Secondary symptoms - Uncoordinated movements, Abortion, Death

2. **Disease name** - Foot-and-mouth disease
   Primary symptoms - Formation of vesicles, blisters in the mouth and feet, lameness
   Secondary symptoms - Lacrimation, Fever, Mycarditis, sudden death

3. **Disease name** - Lumpy-skin disease
   Primary symptoms - Lumpy skin, Emaciation, swollen legs, sores, Pneumonia
   Secondary symptoms - Loss of milk production, Infertility, Abortion, damage to hide

and a list of hypothetical symptoms reported during consultation:

- Fever
- Abortion
- Breathlessness
- Death
- Swollen legs

a carefully constructed set of rules can enable the inference engine to draw conclusions that can be considered to be valid. For each reported symptom, adding the associated disease in an intermediate list stored in the working memory, as described above, allows the system to narrow down the list of probable diseases. A simple rule that can be used to induce this behavior is shown below:

```
Rule Fever
  IF
    Symptom == Fever
  THEN
    pd_1 = new ProbableDisease(Anaplamosis,SymGroup = primary);
    pd_2 = new ProbableDisease(Foot-and-mouth,SymGroup = secondary);
    PossibleDiseases.Add(pd_1);
    PossibleDiseases.Add(pd_2);
```

Listing 5.2: Example rule for executation by hypothetical inference engine

The rule above can be used to specify the behavior of the hypothetical inference engine when `Fever` is one of the symptoms reported by a user during consultation, as

the rule name suggests. The rule states that when the presence of the `Fever` symptom has been confirmed, shown by the line: `Symptom == Fever`, new *probable diseases* should be added into the intermediate list of possible diseases discussed earlier. Probable diseases in this case are diseases to which the `Fever` symptom is associated. Variables `pd_1` and `pd_2` are placeholders for the probable diseases identified. The line `PossibleDiseases.Add(pd_2)` shows how a probable disease would be added into the list of possible diseases kept in the working memory. The addition of a disease in the list involves *1)* the creation of an instance of a hypothetical object used to store a probable disease; *2)* specifying the name of the disease to which the identified symptom is associated, in this case fever, and *3)* specifying the symptom group of the disease in which the identified symptom belongs. These could be achieved by executing the line: `pd_1 = new ProbableDisease(Anaplomosis, SymGroup = primary)` for the *Anaplamosis* disease as shown in the rule above. This would essentially create a *Propable disease* object with the relevant parameters. In this example, it can be seen from the list of diseases above that *Fever* is part of the main symptoms group of the *Anaplamosis* disease, and part of the secondary symptoms group of the *Foot-and-mouth* disease. This is done to establish a weighting system. As primary symptoms are direct results of a pathogen in an animal, these have a higher weighting when compared to their counterparts. As a result, adding both diseases as shown in the consequent of the rule above will result in the *Anaplamosis* disease having a higher weighting in the disease list compared to the *Foot-and-mouth* disease.

To arrive at a diagnosis, the list of possible disease would be analyzed. The disease with the highest frequency and weighting can be concluded as the diagnosis. A high frequency for a given disease would imply that most reported symptoms were associated with the disease. A significantly higher total weighting for a disease would imply that most reported symptoms were associated with the disease. This can also mean that most of the reported symptoms belonged to the primary symptoms group of the disease, thereby making the disease the most likely diagnosis. As frequency increases with an increase in weighting, the disease with the highest weighting will be the disease with the highest frequency, therefore eliminating the possibility of occurrences where the highest frequency and weighting are distributed across two diseases. Given the case above, it can be shown that the approach discussed can allow for valid conclusions to be reached. For this exercise, it will suffice to assume that rules in the knowledge base are similar to the rule discussed above. Shown below is the state of the working memory of the hypothetical inference engine before and after inference using the approach discussed above.

```
Symptoms reported: Fever, Abortion, Breathlessness, Death, Swollen legs
Symptom weighting:  primary = 2 secondary = 1

State before inference:
=======================
Contents of list: [ ]
Diagnosis: null

State after inference:
======================
Contents of list: [(Anaplamosis,primary)(Foot-and-mouth, secondary)(Lumpy-
    skin,secondary) (Anaplamosis,primary)(Anaplamosis,secondary)(Foot-and-
    mouth,secondary)(Lumpy-skin,secondary)]
Diagnosis:  Anaplamosis
```

Listing 5.3: State of working memory before and after inference

The listing above shows the states of the working memory of the proposed inference engine, while Table 5.1 gives the frequency and weighting values for each disease.

Table 5.1: Results of inference step by the inference engine

Disease	Frequency	Weighting
Anaplamosis	3	5
Foot-and-mouth	2	2
Lumpy-skin	2	2

From the illustration of the different states above, it can be seen that the diagnosis reached is justified. Moreover, analyzing the contents of the intermediate list shows that most reported symptoms were associated with the *Anaplamosis* disease, hence the conclusion. From Table 5.1, it can be seen that the *Anaplamosis* disease has the highest frequency and weighting, thereby suggesting that it is the most likely diagnosis.

Although hypothetical and rudimentary, the approach discussed above shows that valid conclusions can be made. The use of a hypothetical inference engine to demonstrate the credibility of a knowledge base is not uncommon; hence, the use of this method in this project.

5.3 Evaluation using VP-Expert shell

It is argued that the use of an expert system shell during development allows for the creation of practical systems [10]. This further allows the developer to separate concerns

and shift focus on developing a specific confined set of components instead of developing every single required component. Chau et al. [10] argued that the employment of run-time evaluation enables the detection of errors at various stages during and after the development process. An integration model is proposed where a confined set of knowledge base files are integrated with an expert system shell to demonstrate the efficiency of run-time evaluation. The author shows that integration is feasible through a series of module, integration and system tests. It is crucial that the different modules work well separately. However, it is equally important that the system as a whole works as specified. In this case, the successful generation of an executable knowledge base is evidence that suggests that the different modules that make up the KAT system function properly and by doing so, enable the system as a whole to fulfill its intended purpose.

A VP-Expert expert system shell was used to investigate the potential of the generated knowledge base to be executed. This required altering the structure and syntax of the generated knowledge base, to allow the VP-Expert shell to recognize the knowledge base. Alteration included restructuring of rules to adhere to the VP-Expert specification, inclusion of control and query statements in the generation of the knowledge base, and changing the file extension of the generated knowledge base file to suit the VP-Expert shell.

5.3.1 Running the VP-Expert shell

The VP-Expert shell was designed to be run on DOS. At start-up, a main menu is displayed allowing the user to select different functions offered. Selecting the `Consult` option invokes the shell to execute the expert system on the current knowledge base. A list of available knowledge base files is displayed. These are stored as knowledge base files (`.KBS`) in the program's directory. Selecting one of the knowledge bases invokes the shell to start a consultation session using the selected knowledge base. Generated knowledge base files were loaded into the shell in an attempt to test the validity of the output of the KAT system.

5.3.2 Test cases

Running the VP-Expert shell using the generated knowledge base file as the knowledge base yielded satisfactory results. In almost all the cases, the conclusion reached was in

accordance to the conclusion in the knowledge source. In cases where an incorrect diagnosis was given, this could be attributed to the limitations inherent in the inference engine used. This is explained in more detail shortly. Discussed below are test cases and the results obtained to show evidence of the credibility of the generated knowledge base.

Three diseases were added into the KAT system.

1. **Disease name** - Anthrax
   Primary symptoms - Sudden death, fever, difficulty breathing, difficulty swallowing
   Secondary symptoms - Muscle tremors, redness of mucous membranes, blood-stained discharge from orifices

2. **Disease name** - Botulism
   Primary symptoms - Weak muscles, immobility, sticking out of tongue, salivation before death
   Secondary symptoms - Severe illness, stiff and slow walking, rough coat, thinness, sunken eyes

3. **Disease name** - Brucellosis
   Primary symptoms - Abortion in late pregnancy, infertile cows, infertile bulls
   Secondary symptoms - Swelling of testicles, swollen joints

Table 5.2 shows the distribution of symptoms across diseases. It should be noted that the character P denotes a primary symptom and the character S denotes a secondary symptom.

Rules were generated using the disease information stored in the system. The resulting knowledge base was loaded *as is* into the VP-Expert shell. Sets of hypothetical symptoms were supplied to the expert system during querying. After inference, the conclusion or diagnosis made in each case was noted. The diagnoses reached were found to be accurate, hence validating the knowledge base generated by the KAT system. Discussed below are the results for the different cases:

Case 1

Listing 5.4 shows the prompts used by VP-Expert shell to query the user for symptoms. The answers supplied by the user were matched against the rules in the knowledge base,

Table 5.2: Distribution of symptoms across diseases.

Symptom	Anthrax	Botulism	Brucellosis
Abortion in late pregrancy			P
Blood-stained discharge	S		
Difficult breathing	S		
Difficulty swallowing	S		
Fever	P		
Immobility		P	
Infertile bulls			P
Infertile cows			P
Muscle tremors	S		
Redness of mucous membranes	S		
Rough coat		S	
Salivation before death		P	
Severe illness		S	
Sticking out of tongue		P	
Stiff and slow walking		S	
Sudden death	P		
Sunken eyes		S	
Swelling of testicles			S
Swollen joints			S
Thinness		S	
Weak muscles		P	

allowing the expert system to reach a conclusion.

```
Any signs of sudden death? >Yes No
Is there fever? Yes >No
Any signs of abortion in late pregnancy? >Yes No
Is there infertile cows? Yes >No
Any signs of weak muscles? >Yes No
Any immobility? Yes >No
Any muscles tremors? Yes >No
Is there redness of mucous membranes? >Yes No
Is there blood-stained discharge from orifices? Yes >No
Any difficult breathing? Yes >No
Any difficulty swallowing? Yes >No
Any signs of swelling of throat and neck? Yes >No
Any signs of infertile bulls? Yes >No
Any sticking out of tongue? Yes >No
Any signs of salivation before death? Yes >No
```

Listing 5.4: Querying of symptoms by VP-Expert

It can be seen from the information supplied above that the user reported symptoms: **sudden death, abortions in late pregnancy, weak muscles, and redness of mucous membranes**. Additionally, it can be seen from the table showing the distribution of symptoms that two of the symptoms listed belong to the *Anthrax* disease, one to the *Brucellosis* disease, and a different one to the *Botulism* disease. The most likely diagnosis in this case would be *Anthrax*. This is because the highest number of reported symptoms from a single disease are associated with the *Anthrax* disease. It can be argued that a better approach could be used, however, the rule types were constructed to allow for this kind of behavior. Shown below is a screenshot of the VP-Expert shell at the end of a consultation session. As expected, the expert system reported *Anthrax* as the most likely diagnosis.

Figure 5.2: Screenshot of VP-Expert interface showing an Anthrax diagnosis

The topmost green rectangular section of the screen shows the conclusion reached by the expert system. The disease name and treatment suggested are shown as the inference step concludes. This result contributes to the overall credibility of the generated knowledge base.

Case 2

A second case presents a different set of symptoms presented to the expert system.

```
Any sudden death? Yes >No
Is there abortion in late pregnancy? Yes >No
Is there weak muscles? >Yes No
Any immobility? >Yes No
Any sticking out of tongue? >Yes No
Any signs of salivation before death? >Yes No
```

Listing 5.5: Querying of symptoms by the expert system in the second test case

In this example, the user reported symptoms the following symptoms: **weak muscles, immobility, sticking out of tongue, and salivation before death**. From the table of symptoms, it can be seen that these are the primary symptoms of the *Botulism* disease. Consequently, a diagnosis of the disease *Botulism* is expected. which is exactly the diagnosis given by the expert system.

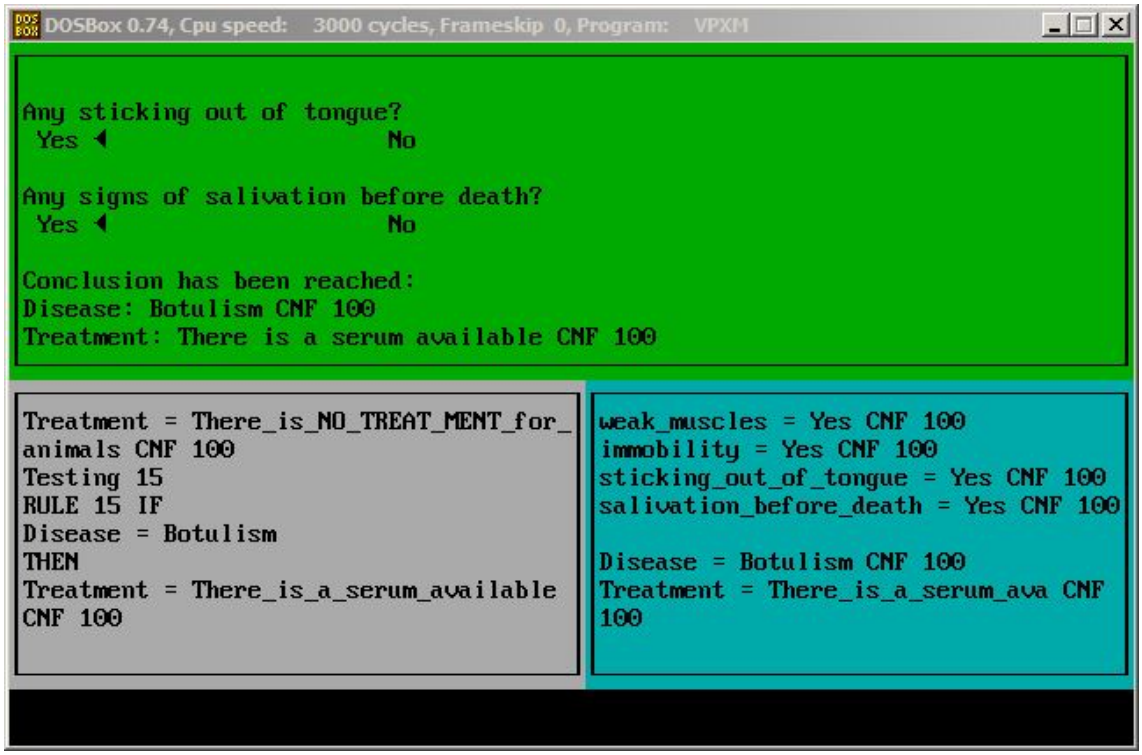

Figure 5.3: Screenshot of VP-Expert interface showing an Botulism diagnosis

A diagnosis together with the suggested treatment is given at the end of the inference step. It is worth stating that the VP-Expert shell places a limit on the number of characters

in strings used in the knowledge base. For this reason, string values have been kept short. On an ideal system, more informative descriptions of diagnoses would be provided, however, this is not possible using the VP-Expert shell.

Case 3

A third and final case is presented to demonstrate the validity of the knowledge base. A small set of symptoms was presented to the expert system.

```
Any sudden death? >Yes No
Any fever? Yes >No
Is there abortion in late pregnancy? >Yes No
Is there infertile cows? Yes >No
Is there weak muscles? Yes >No
Any muscles tremors? Yes >No
Is there redness of mucous membranes? Yes >No
Is there blood-stained stained discharge from orifices? Yes >No
Any difficult breathing? Yes >No
Any difficulty swallowing? Yes >No
Any signs of swelling of throat and neck? Yes >No
Any signs of infertile bulls? Yes >No
Any immobility? Yes >No
Any sticking out of tongue? Yes >No
Any signs of salivation before death? Yes >No
Any severe illness? Yes >No
Any rough coat? Yes >No
Is there thinness? Yes >No
Is there sunken eyes? Yes >No
```

Listing 5.6: Querying of symptoms by the expert system in the third test case

In this case, only two symptoms were reported: **sudden death and abortion in late pregnancy**. This example demonstrates the expected functionality of the expert system shell. As a small amount of information is supplied by the user, the shell proceeds to prompt the user for more symptoms in an attempt to gather enough information. This is done regardless of the fact that the two symptoms were supplied fairly early on during the querying of symptoms. In any case, an interesting diagnosis is reached, or rather none is reached.

The *Undecided* conclusion can be argued to be a satisfactory one. Since two primary symptoms of two different diseases are supplied, the expert system can either conclude

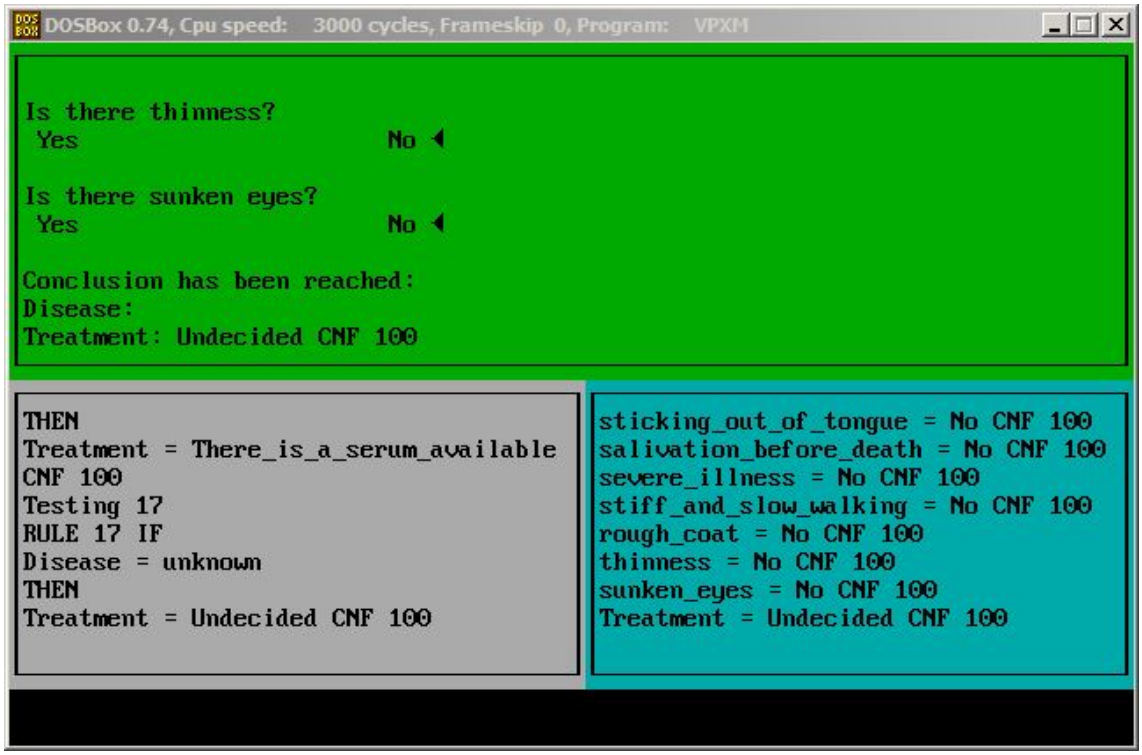

Figure 5.4: Screenshot of VP-Expert interface showing an Undecided conclusion

that both diseases are probable or give an undecided conclusion. The latter is in fact the case in this example.

5.3.3 Limitations of VP-Expert expert system shell

The VP-Expert expert system shell was developed and released some decades ago by Paperback Software International. It soon rose into popularity in both educational and commercial fields. As mentioned earlier, two genres of the software package are available: a commercial version, and a student version. The student version was used for the evaluation of the generated knowledge base. As a technology that dates back more than two decades ago, the VP-Expert shell presented some limitations during the evaluation of the generated knowledge base. These presented challenges during the evaluation process. The first of these is the inability prioritize or choose between conflicting conclusions, that is, no strategies have been put in place to allow preference of one conclusion over another in the occurrence of a clash. In this case, unless specified in the knowledge base, VP-Expert finds it difficult to make a conclusion if no additional information is provided to help differentiate the conflicting diseases. In addition, the short-circuit evaluation employed by the shell was found to present difficulties in some cases, which ultimately

led to incorrect conclusions. An example of these is the case where the shell prematurely concludes the consultation session once some of the primary symptoms of a given disease have been matched, thereby neglecting the possibility of reaching a different diagnosis had all symptoms been queried. For some of these limitations, alterations in the knowledge base were enough to eliminate the problem. Over and above that, as mentioned earlier, the VP-Expert shell places a limit on the length of character sequences or text values used in the knowledge base. This hindered the provision of more informative descriptions as well as the construction of more complex rules. Consequently, several of the incorrect diagnosis reached during testing could be attributed to some of the limitations presented by the VP-Expert shell.

5.4 Discussion

As noted in Chapter 3, the evaluation of the generated rule base was seen as a critical step in the overall process of knowledge base generation. This presented a means to verify and validate the resulting rule base. It was shown that by combining the functionality and output of the different modules that make up the KAT system, an executable knowledge base could be generated. The generated knowledge base was found to meet the success criteria given earlier in this chapter. The potential to evaluate the knowledge base in more ways than one highlights the flexibility and extensibility of the system and the output thereof. It was shown that by simply making minor changes to the structure and syntax of the generated knowledge base file, the output of the system could be tailored to different executing programs, thereby allowing the knowledge base to be evaluated on multiple systems. Over and above, there is the need to highlight the indivisible nature of an expert system, which was found to present difficulties during the development of the rule generator. The two main components of an expert system, namely, the inference engine and the knowledge base, are designed to work closely together to achieve a common goal. As a result, this requires that both components are able to interact seamlessly with one another in an attempt to establish cohesion. This in turn requires that the development of one component happens in conjunction with the development of another. Attempting to construct an executable knowledge base in isolation stimulated much thought on the functionality of a suitable executing program. This notably highlighted the indivisible nature of an expert system.

5.5 Summary

This chapter discussed the evaluation of the generated knowledge base. First, success criteria were presented. Next, manual evaluation of the knowledge base was discussed with the discussion focusing on the structure and functionality of a hypothetical inference engine believed to be suitable for the execution of the generated knowledge base. A discussion of the run-time evaluation using a VP-Expert expert system shell was given. Test cases were used to demonstrate the credibility of the knowledge base. Additionally, the limitations presented by the expert system shell used were briefly discussed. Finally, a discussion on the overall evaluation process was given.

Chapter 6

Conclusion

The research project was centered around the knowledge engineering process in expert system development. This involved the investigation of the use of different knowledge base construction tools available, research on various types and examples of expert systems, their domains, and the knowledge representation schemes used to represent knowledge. The project's primary aim was to present an efficient approach to knowledge representation in the development of expert systems by developing a modularized tool to enable the automated construction of knowledge bases.

The design and implementation of the system was discussed with emphasis on the functionality of the different components that together enabled the generation of an executable knowledge base. The developed system was tested and was found to meet the project objectives.

6.1 Research objectives

As noted, the project aimed to investigate the possibility of automating the knowledge engineering process in the development of expert systems. This would enable the automated construction of an executable knowledge base. By doing so, the role of the knowledge engineer would be assumed by the developed tool. Additionally, the developed tool would make possible extension of a knowledge base by domain experts. It was required that the generated knowledge base *1)* successfully represents knowledge as rules; *2)* is executable by a suitable inference engine, and *3)* allows for valid conclusions to be reached.

It was shown that the developed system meets the defined objects. The generated knowledge base was found to represent the domain knowledge contained in the knowledge sources. Furthermore, the developed tool allowed the effortless extension of the generated knowledge base by domain experts. Additionally, it was shown that by introducing minor changes to the structure of the generated knowledge base, the knowledge base could be tailored to different executing programs, hence, confirming the knowledge base executable. Over and above, it was shown that valid conclusions could be reached from the execution of the generated knowledge base.

6.2 Future work

Despite meeting the project objectives, areas of improvement were identified during the development and evaluation of the KAT system.

- Currently, the developed tool implements a rule-based knowledge representation scheme. Implementing additional alternative schemes would significantly extend and improve the functionality of the tool. This could allow the use of a hybrid representation scheme, thereby improving the credibility of the generated knowledge base. Alternative representation forms have been provided for on the GUI.
- The KAT system makes use of `Textfield` controls to facilitate the entry of new knowledge into the system. Implementing alternative ways of data input could improve the usability of the system. Alternative ways include input from a text file, or from an online source.
- To improve the quality and credibility of the generation of knowledge base, a dynamic approach could be taken during the elicitation process. By using information contained in the knowledge base to facilitate the elicitation process, the system could elicit from the domain expert additional items of information that could help supplement the knowledge already obtained, thereby enabling the generation of a more complex and comprehensive knowledge base.
- Introducing confidence factors in the rules generated could improve the inference process during execution of the knowledge base by executing programs.

Bibliography

[1] Abdullahi, M., Al-Mattarneh, H. M. A., Hassan, A. H., Hassan, M. A., and Mohammed, B. S. A review on expert systems for concrete mix design. *The International Conference on Construction and Building Technology 21* (2008), 231238.

[2] ADELI, H. Insight 2+. *Computer-Aided Civil and Infrastructure Engineering 2*, 2 (1987), 173–174.

[3] Akram, M., Rahman, I. A., and Memon, I. A review on expert system and its applications in civil engineering. *International Journal of Civil Engineering and Built Enviroment 1*, 1 (2014), 24–29.

[4] Arnold, K., and Gosling, J. *The Java Programming Language*. Addison-Wesley, 1998.

[5] Beckert, B. Introduction to Artificial Intelligence: First-order Logic. Online, 2005. Accessed on: May 20, 2014. Available from: `http://www.skit.edu.in/menu/CSE/r09/AI/08FirstOrderLogic.pdf`.

[6] Bennett, J. S. ROGET: a knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning 1* (1985), 49–74.

[7] Boose, J. H. A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition 1* (1989), 3–37.

[8] Brachman, R. J., and Levesque, H. J. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2004.

[9] Chandana, S., Mayorga, R. V., and Chan, C. W. Automated knowledge engineering. *Engineering and Technology 39* (2008), 511–520.

[10] Chau, K. W., and Yang, W.-W. Structuring and evaluation of vp-expert based knowledge bases. *Engineering Applications of Artificial Intelligence 7* (1994), 447–454.

[11] Dewanto, Satrio, and Lukas, Jonathan. Expert system for diagnosis pest and disease in fruit plants. *EPJ Web of Conferences 68* (2014), 00024.

[12] Diederich, J., and Rumann, I. KRITON: a knowledge-acquisition tool for expert systems. *Int. J. Man-Machine Studies 26* (1987), 29–40.

[13] Duda, R. O., and Shortlife, E. H. Expert systems resarch. *Science 220* (1983), 261–268.

[14] Exsys Inc. Exsys Corvid Expert System Development Tool. Online, 2011. Accessed on: October 15, 2014. Available from: `http://www.exsys.com/exsyscorvid.html`.

[15] Ford, K., Canas, A., Jones, J., Stahl, H., Novak, J., and Adams-Webber, J. ICONKAT: an integrated constructivist knowledge acquisition tool. *Knowledge Acquisition 3*, 2 (1991), 215 – 236.

[16] Frederick, H.-R., Waterman, D., and Lenat, D. *Building Expert Systems.* Addison-Wesley Longman Publishing Co., Inc., 1983.

[17] Friederich, S., and Gargano, M. *Expert Systems Design and Development Using VP-Expert.* Wiley, 1989.

[18] HTML.net. HTML. Online. Accessed on: May 20, 2014. Available from: `http://html.net/`.

[19] Iancu, E., Mates, D., and Voicu, V. Considerations regarding the expert systems in the economy and the use method of the production systems based on rules. *Journal of Applied Computer Science & Mathematics 1* (2010), 63–65.

[20] Japanese Technology Evaluation Center. The Applications of Expert Systems. Online, May 1993. Accessed on: May 20, 2014. Available from: `http://www.wtec.org/loyola/kb/c1_s2.htm`.

[21] Kaetzel, L., and Clifton, J. R. Expert/knowledge based systems for materials in the construction industry: State-of-the-art report. *Materials and Structures 28* (1995), 160–174.

[22] Kahn, G., Nowlan, S., and McDermott, J. MORE: An intelligent knowledge acquisition tool. In *Proceedings of the Ninth International Joint Conference on Artificial* (1985).

[23] Kurmangaziyeva, L. T., Utenova, B. E., and Mailybayeva, A. J. Expert systems and their use in oil and gas complex. *Life Science Journal 11* (2014), 392–395.

[24] Martin, J., and Oxman, S. *Building Expert Systems: A tutorial*. Prentice-Hall, 1988.

[25] Matsatsinis, N., Doumpos, M., and Zopounidis, C. Knowledge acquisition and representation for expert systems in the field of financial analysis. *Expert Systems with Applications 12*, 2 (1997), 247 – 262.

[26] McCarthy, J. *LISP 1.5 Programmer's Manual*. MIT Press, 1965.

[27] McDermott, J. R1: A rule-based configurer of computer systems. *Artificial Intelligence 19* (1982), 39–88.

[28] McNamee, P., Mayfield, J., Finin, T., Oates, T., Lawrie, D., Xu, T., and Oard, D. KELVIN: a tool for automated knowledge base construction. In *Proc. Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (June 2013), Association for Computational Linguistics. (demonstration paper).

[29] Microsoft. ASP.NET. Online. Accessed on: May 20, 2014. Available from: `http://www.asp.net/`.

[30] Microsoft. Introduction to the C# Language and the .NET Framework. Online. Accessed on: October 15, 2014. Available from: `http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx`.

[31] Microsoft. Windows Forms. Online. Accessed on: October 15, 2014. Available from: `http://msdn.microsoft.com/en-us/library/dd30h2yb(v=vs.110).aspx`.

[32] Netscape Communications Corporation. JavaScript. Online. Accessed on: May 20, 2014. Available from: `http://www.javascriptkit.com/`.

[33] Niwa, K., Sasaki, K., and Ihara, H. An experimental comparison of knowledge representation schemes. *The AI Magazine 5* (1984), 29–36.

[34] Oncontext. First Order Predicate Calculus. Online. Accessed on: May 30, 2014. Available from: `http://www.ontotext.com/factforge/first-order-predicate-calculus`.

[35] Pop, D., and Negru, V. An extensible environment for expert system development. In *Knowledge-Based Intelligent Information and Engineering Systems*, V. Palade, R. Howlett, and L. Jain, Eds., vol. 2773 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2003, pp. 1016–1022.

[36] Rafea, A., and Hazman, H. H. M. Automatic knowledge acquisition tool for irrigation and fertilization expert systems. *Expert Systems with Applications 24* (2003), 49–57.

[37] Rouse, M. First-order Logic. Online, September 2005. Accessed on: May 20, 2014. Available from: `http://whatis.techtarget.com/definition/first-order-logic`.

[38] Sawyer, B., and Foster, D. L. *Programming Expert Systems in Pascal*. Wiley Press, 1986.

[39] Shortlife, E. H. *Computer-Based Medical Consultations: MYCIN*. Elsevier, 1976.

[40] Sowa, J. F. *Principles of Semantic Networks*. Morgan Kaufmann, 1991.

[41] Tanwar, P., Prasad, D. T., and Datta, D. K. Hybrid technique for effective knowledge representation & a comparative study. *International Journal of Computer Science & Engineering Survey 3* (2012), No. 4.

[42] ten Berge, T., and van Hezewijk, R. Procedural and declarative knowledge. *Theory & Psychology 9* (1999), 605–624.

[43] van de Gevel, A. J. W., and Noussair, C. N. *The Nexus Between Artificial Intelligence and Economics*. Springer, 2013.

[44] w3Schools.com. XML. Online. Accessed on: May 20, 2014. Available from: `http://www.w3schools.com/xml/default.ASP`.

[45] Walden, M. Sports Injury Clinic. Online, 2000. Accessed on: May 20, 2014. Available from: `http://www.sportsinjuryclinic.net/`.